2009

RECONOCIMIENTO DE OBJETOS EN UNA COCINA CON UNA WEBCAM

OpenCV

Computer Vision with the OpenCV Library

Proyecto Fin de Carrera

Universidad Carlos III de Madrid

Departamento de Informática

Alumno: Francisco Javier García Fernández

Tutor: Javier Ortiz Laguna

30/11/2009

ÍNDICE

CAPÍTULO I

Introducción

1.1 INTRODUCCIÓN

Este proyecto fin de carrera tiene como finalidad formar parte de un sistema de reconocimiento de actividades. Así pues, la meta final del sistema del que va a formar parte es reconocer las actividades que una persona realiza en un entorno determinado, en este caso una cocina. Para ello, el sistema a desarrollar en este proyecto pretende explorar el uso de una cámara web con el objetivo de reconocer los objetos que la persona utiliza en el entorno escogido. Dichos objetos podrían servir de ayuda para reconocer las acciones que el usuario está llevando a cabo.

El reconocimiento de actividades tiene como objetivo reconocer las diferentes acciones que puede llevar a cabo un ser humano mediante la observación de diferentes factores. Pongamos el caso de una persona que va a beber agua. El propio movimiento de sus brazos, cabeza, boca, etc. nos puede indicar la acción que está realizando pero también puede ayudar significativamente saber que esta persona tiene una botella entre sus manos y que esta se mueve hacia la cabeza. Por tanto, entre los factores más importantes a observar podemos incluir los movimientos de las personas, el reconocimiento y seguimiento de los objetos presentes en el espacio de trabajo y los cambios que se producen en el entorno como por ejemplo: la aparición de humo, la apertura de una puerta, etc.

La meta de este proyecto es el reconocimiento y seguimiento de los objetos que se encuentran en el espacio de trabajo que como hemos visto es algo fundamental en el reconocimiento de actividades. La ciencia que se encarga de estas funciones es la visión por computador.

La visión por computador o visión artificial es un subcampo de la inteligencia artificial (IA). La visión por computador es la ciencia que desarrolla la base teórica y algorítmica mediante la que se extrae y analiza información sobre el mundo o el entorno a través de una imagen o una secuencia de imágenes. El propósito de la visión artificial es programar un computador para que "entienda" una escena o las características de una imagen, imitando de alguna manera el sistema visual humano. Es decir, responder a dos preguntas fundamentales: ¿qué objetos hay y dónde se encuentran?

Para reconocer los objetos que hay en la imagen y saber de qué tipo son, la aplicación que vamos a crear necesita haber sido entrenada con anterioridad. Es decir, cuando una persona ve una mesa, sabe que es una mesa porque la ha visto en ocasiones anteriores y alguien le ha dicho que se trata de ese objeto. De alguna manera con el programa pasa algo similar, hay que mostrarle varias imágenes de los objetos y decirle cuál es su nombre y sus características

principales. Por ello se ha creado una base de datos de imágenes y un fichero que indica a la aplicación los datos necesarios para poder entrenar la aplicación.

A continuación se detallarán los objetivos principales del proyecto y se hará un breve resumen del contenido de la memoria.

1.2 OBJETIVOS INICIALES DEL PROYECTO

El proyecto tiene como objetivo el desarrollo de una aplicación de visión por computador con las siguientes características:

- Capacidad para detectar una serie de objetos en una cocina y reconocer el tipo de objetos que son.
- Seguir los movimientos de estos objetos.
- La captura de las imágenes se lleva a cabo a través de una webcam y el funcionamiento de la aplicación es en tiempo real.
- Debe tener una opción para poder trabajar con vídeos.
- Posibilidad de modificar algunos parámetros de la aplicación a través de un fichero de configuración.
- Debe crear un fichero de salida en el que se indique en qué posición se encuentran los objetos en cada frame.
- La aplicación se debe desarrollar bajo el uso de software libre y sobre un entorno de programación multiplataforma.

1.3 CONTENIDO DE LA MEMORIA

En este apartado se describe brevemente el contenido de los diferentes apartados de la memoria:

Capítulo I: Introducción

En este apartado se explican de forma introductoria los objetivos y características principales del proyecto, así como un resumen del contenido de la memoria.

Capítulo II: Estado del arte

En este capítulo, como su nombre indica, se dan a conocer las bases teóricas sobre las que se fundamenta el proyecto. No sólo se describen las técnicas que se han usado para el desarrollo del proyecto, también aquellas técnicas que por su importancia y utilización dentro del campo de la visión artificial he creído conveniente destacar.

Capítulo III: Materiales empleados

Este apartado trata sobre los elementos hardware y software que se han utilizado durante el desarrollo del proyecto, así como la descripción del entorno de desarrollo. En esta descripción se analizan los motivos por los que se ha decidido usar dicho entorno de desarrollo.

Capítulo IV: Metodología empleada

Descripción de las técnicas usadas en el desarrollo de la aplicación. Se detallan los procedimientos seguidos para segmentar y procesar imágenes, describir y representar los objetos, clasificarlos y posteriormente seguir su trayectoria.

Capítulo V: Resultados obtenidos

En este capítulo se analizan los resultados obtenidos, justificando si se han cumplido los objetivos iniciales el proyecto y proponiendo futuras mejoras del sistema.

Capítulo VI: Bibliografía

Este apartado nos presenta las referencias que se han consultado para el desarrollo del documento del proyecto.

Anexo A: Manual de usuario

El manual de usuario es un documento técnico de la aplicación que intenta dar asistencia a sus posibles usuarios. El manual de usuario está escrito de tal manera que debería poder ser entendido por cualquier usuario principiante y también serle útil a usuarios avanzados.

CAPÍTULO II

Estado del arte

2.1 VISIÓN POR COMPUTADOR

En este capítulo se presenta al lector el área de investigación donde se enmarca este proyecto, dentro del amplio espectro de posibles áreas que engloban las ciencias de la computación. A continuación se lleva a cabo un breve recorrido por la historia de la visión por computador, para profundizar posteriormente en el análisis y procesamiento de imágenes y en el reconocimiento de objetos. Especialidad donde se enmarca exactamente el proyecto realizado.

2.1.1 HISTORIA DE LA VISIÓN POR COMPUTADOR

Desde los inicios de la informática, se ha perseguido la replicación del comportamiento humano en las máquinas. El cuerpo humano ha sido y es considerado como el modelo más eficiente a seguir para determinadas disciplinas. Por ejemplo: la robótica, la visión por computador, el aprendizaje basado en la experiencia, y un largo etcétera de disciplinas incluidas en la informática.

El problema de la elección de dicho modelo como patrón, resulta en consecuencias frustrantes para cualquier línea de investigación. Hoy por hoy, comportamientos triviales para el hombre no están todavía científicamente tipificados, y entre ellos el proceso de la visión humana. Por esta razón la construcción de un sistema que emule el sistema visual humano sería prácticamente imposible. Aunque existe una enorme cantidad de publicaciones en neurofisiología, psicología y psicofísica lo que se conoce del sistema de visión humano más allá del propio ojo es principalmente disjunto, especulativo y escaso.

Todo esto originó el abandono paulatino de esta línea de investigación, hasta que en la década de los ochenta se produce un replanteamiento en la especificación de los objetivos para esta disciplina. Se cambia la ambiciosa denominación de "Visión Artificial", por una más humilde y consecuente con el objetivo perseguido, como es "Visión por Computador". A este cambio en el punto de vista sobre esta disciplina se le unen, la existencia de computadores más potentes, elementos hardware específicos que relevan el uso de algoritmos complejos, y una mayor experiencia histórica en otros campos de la informática. Empiezan a describirse metodologías que dividen el problema de la visión por

computador en distintas fases, cuya solución resulta más asequible, y lo relacionan con otras disciplinas, que hasta el momento eran independientes como pueden ser el procesamiento de imágenes, o el reconocimiento de patrones.

La visión por computador actualmente comprende tanto la obtención como la caracterización e interpretación de las imágenes. Esto supone algoritmos de muy diversos tipos y complejidades. En un sistema de visión por computador actual se pueden distinguir seis etapas o fases:

- **Captación:** Es el proceso a través del cual se obtiene una imagen visual.
- **Preprocesamiento:** Incluye técnicas tales como la reducción de ruido y realce de detalles.
- **Segmentación:** Es el proceso que divide a una imagen en objetos que sean de nuestro interés.
- **Descripción:** Es el proceso mediante el cual se obtienen características convenientes para diferenciar un tipo de objeto de otro, por ejemplo tamaño y forma.
- **Reconocimiento:** Es el proceso que identifica a los objetos de una escena. Por ejemplo las diferentes tipos de piezas en un tablero de ajedrez.
- **Interpretación:** Es el proceso que asocia un significado a un conjunto de objetos reconocidos.

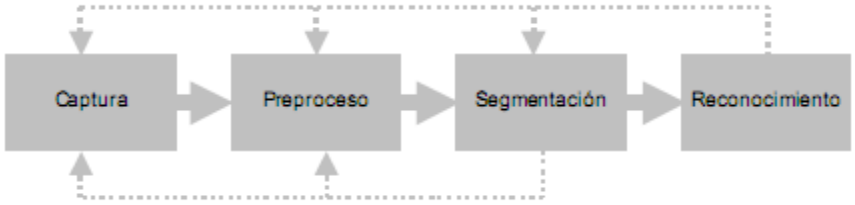


Figura 1.- Esquema general de un sistema de visión por computador.

Esto supone distintos tipos de procesamiento en función del nivel en el que nos movamos:

- **Visión de bajo nivel:** comprende la captación y el preprocesamiento. Ejecuta algoritmos típicamente de filtrado, restauración de la imagen, realce, extracción de contornos, etc.
- **Visión de nivel intermedio:** comprende la segmentación, descripción y reconocimiento, con algoritmos típicamente de extracción de características, reconocimiento de forma y etiquetado de éstas.

- **Visión de alto nivel:** comprende la fase de interpretación, normalmente estos algoritmos se refieren a la interpretación de los datos generalmente mediante procedimientos típicos de la Inteligencia Artificial para acceso a bases de datos, búsquedas, razonamientos aproximados, etc.

Por último cabe destacar el rango de aplicaciones en la que la visión por computador tiene cada vez más, un papel importante:

- **Militares:** gran parte de los logros informáticos conseguidos han sido promovidos o posteriormente adquiridos y mejorados por este sector. Entre las aplicaciones más comunes podemos destacar la detección y seguimiento de objetivos.
- **Robótica:** en aplicaciones industriales para guiado de robots.
- **Análisis de imágenes tomadas por satélite.**
- **Identificación automática de huellas:** muy extendidos en sistemas de seguridad y control de acceso.
- **Control de calidad:** muy extendido en cadenas de montaje.
- **Medicina:** dando especial importancia a los avances obtenidos con imágenes RMI (imágenes por resonancia magnética), y a los ya conocidos logros en imágenes por rayos X.
- **Reconocimiento de Caracteres:** dentro de esta área se encontrarían aplicaciones como lectura de etiquetas, procesamiento de cheques bancarios, lectura de texto, etc.
- **Cartografía: e**laboración de mapas a partir de fotografías, síntesis de mapas de clima, etc.

En definitiva, cada día surge una nueva aplicación de la visión por computador aplicada a un sector que obtiene grandes beneficios de tal aportación.

CONFIGURACIÓN INFORMÁTICA DE UN SISTEMA DE VISIÓN ARTIFICIAL

Aunque se pueden proponer configuraciones muy avanzadas, por ejemplo incluyendo hardware específico para acelerar ciertas operaciones, los elementos imprescindibles son:

- Un **sensor óptico** para captar la imagen: Una cámara de vídeo, una cámara fotográfica, una cámara digital, un escáner... uniéndole un conversor analógico-digital cuando sea preciso.
- Un **computador** que almacene las imágenes y que ejecute los algoritmos de preprocesado, segmentación y reconocimiento de la misma.

2.1.2 ADQUISICIÓN Y REPRESENTACIÓN DE LA IMAGEN DIGITAL

Este capítulo trata los aspectos más relevantes del proceso de captura y digitalización de una imagen, esto es, la adquisición de la imagen del mundo físico y su paso al dominio discreto y virtual informático.

Una vez digitalizada una imagen bidimensional digital está constituida por un conjunto de elementos llamados píxeles. Cada píxel ofrece cierta información sobre una región elemental de la imagen. En imágenes en niveles de gris esta información es el brillo. En imágenes en color, la información corresponde a la intensidad de cada una de las componentes de una base de color (por ejemplo RGB). Dentro de este capítulo también se repasan las técnicas de compresión, que buscan la forma más eficiente de almacenar las imágenes digitales.

CAPTURA Y DIGITALIZACIÓN DE IMÁGENES

Las imágenes digitales son "señales" discretas, que suelen tener origen en una "señal" continua. Por ejemplo, una cámara digital toma imágenes del mundo real que es continuo (tanto el espacio, como el espectro de la radiación reflejada por los objetos se consideran continuos); otro ejemplo es el de un escáner, el cual digitaliza imágenes procedentes de documentos o fotografías que a efectos prácticos también se consideran continuos.

En el proceso de obtención de imágenes digitales se distinguen dos etapas.

La primera, conocida como captura, utiliza un dispositivo, generalmente óptico, con el que obtiene información relativa a una escena. En la segunda etapa, que se conoce como digitalización, se transforma esa información, que es una señal con una o varias componentes continuas, en la imagen digital, que es una señal con todas sus componentes discretas.

MODELOS DE CAPTURA DE IMÁGENES

A grandes rasgos, para capturar una imagen se suele distinguir entre dispositivos pasivos (basados generalmente en el principio de *cámara oscura*) y dispositivos activos (basados en el escaneo). Esta clasificación no incluye todas las formas posibles de creación de imágenes, como por ejemplo la construcción de imágenes sintéticas.

LA CÁMARA OSCURA

El principio de cámara oscura se puede usar para capturar imágenes de escenas tridimensionales (del mundo real) y proyectarlas en un plano bidimensional. Dispositivos de este tipo son las cámaras fotográficas y las cámaras de vídeo. Este modelo además se puede usar para capturar imágenes de elementos bidimensionales, como fotografías y documentos, como por ejemplo hacen los escáneres de cámara. También se pueden usar dos o más cámaras para capturar diferentes perspectivas de una misma escena y construir una representación 3D de la misma.

EL ESCANEO

Este esquema es fundamentalmente distinto del basado en cámara, ya que existe un elemento activo (generalmente un haz de luz láser) que recorre la escena que se desea capturar. Por tanto son imprescindibles dos dispositivos, uno emisor del haz de luz y otro el receptor. El escáner emite el haz de luz y éste, tras chocar con la imagen que se escanea, es recogido en el detector de luz. Repitiendo este proceso de manera continua se puede construir una señal que corresponde a una representación de la escena.

Los dispositivos basados en el escaneo también se usan con diferentes fines. Así, los escáneres-láser pueden capturar escenas 3D directamente, y los escáneres de tambor permiten capturar imágenes de elementos bidimensionales.

Los dispositivos basados en cámara aventajan a los basados en escaneo en velocidad. Además son más simples y se parecen más al sistema visual humano. Es de prever que, con el tiempo, los modelos de cámara terminen superando también a los de escaneo en cuanto a calidad de la imagen obtenida, ya que su principal cuello de botella, que se encuentra actualmente en el elemento digitalizador, parece que puede ser mejorado sensiblemente con nuevos desarrollos tecnológicos.

DIGITALIZACIÓN

Es el proceso de paso del mundo continuo (o analógico) al mundo discreto (o digital). En la digitalización normalmente se distinguen dos procesos: el muestreo (“sampling”) y la cuantificación (“quantization”).

MUESTREO

El muestreo de una señal continua consiste en la medición a intervalos (discretización) respecto de alguna variable (generalmente el tiempo o el espacio), siendo su parámetro fundamental la *frecuencia de muestreo*, que representa el número de veces que se mide un valor analógico por unidad de cambio.

Mediante el muestreo se convierte una imagen IC, que es algo continuo, en una matriz discreta ID de N×M píxeles. El número de muestras por unidad de espacio sobre el objeto original conduce al concepto de *resolución espacial* de la imagen. Ésta se define como la distancia, sobre el objeto original, entre dos píxeles adyacentes. Sin embargo la unidad de medida de resolución espacial más habitual suele ser los píxeles por pulgada (comúnmente DPIs) siempre medidos sobre el objeto original.

CUANTIFICACIÓN

La segunda operación es la cuantificación de la señal, que consiste en la discretización de los posibles valores de cada píxel. Los niveles de cuantificación suelen ser potencias de 2 para facilitar el almacenamiento en el computador de las imágenes, ya que éstos utilizan el byte como unidad mínima de memoria directamente direccionable. Así, suelen usarse 2, 4, 16 ó 256 niveles posibles. De esta forma, ID que pertenece a $\Re$ se convierte en IDC (discreta cuantificada) que pertenece a N.

REPRESENTACIÓN DE LA IMAGEN Y ESTRUCTURAS DE DATOS

Las imágenes suelen almacenarse en los ordenadores en forma de ficheros. En este punto se analizarán las estructuras que se usan a tal efecto, los métodos utilizados para optimizar el espacio requerido y algunos de los diferentes formatos estándar (TIFF, GIF, BMP, JPG...).

ESTRUCTURA DEL FICHERO DE IMAGEN

Generalmente una imagen almacenada en un ordenador está constituida (ver Figura 2) por un mapa de bits (sería mejor decir de píxeles) precedido por una cabecera que describe sus características (tamaño de la imagen, modo de color, paleta, resolución de la imagen...). Frecuentemente, cuando la imagen se encuentra en la memoria principal del ordenador la cabecera y el mapa de bits no están contiguos.

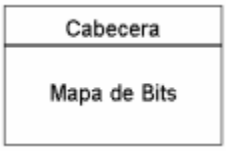


Figura 2.- Esquema de una imagen en fichero

COMPRESIÓN DE IMÁGENES

En ocasiones es impracticable el tratamiento directo de ciertas imágenes debido a la gran cantidad de datos que requiere su almacenamiento o su transmisión. La

compresión de las imágenes trata este problema, mediante la reducción de la cantidad de datos necesarios para representar una imagen digital, esta reducción de datos se realiza porque existe cierta información redundante, es decir, que no es necesaria o es menos importante.

Clásicamente se distinguen tres tipos de redundancia:

REDUNDANCIA EN LA CODIFICACIÓN

Hasta ahora se ha usado un tamaño fijo para representar la información de cada punto de una imagen. Por ejemplo, se ha usado un byte para representar la intensidad de cada punto de una imagen con un nivel de cuantificación de 256 niveles de gris.

Sin embargo en una imagen suelen existir niveles de intensidad que son más probables que otros porque aparecen más veces. La codificación de tamaño variable es una técnica de compresión que aprovecha esta circunstancia. Consiste en asignarle un código más corto a los niveles de intensidad más probables (que aparecen más veces) y más largo a los menos probables (que aparecen menos veces), consiguiendo una reducción del tamaño de los datos de la imagen. Algunos ejemplos de este tipo de compresión son el método de *compresión Huffman* y el algoritmo de compresión *LZW*.

REDUNDANCIA EN LA REPRESENTACIÓN ESPACIAL DE LOS PÍXELES

Una figura regular ofrece una alta correlación entre sus píxeles. Esta correlación da lugar a una redundancia espacial o geométrica si la forma de representación no es la adecuada. La forma más usual de tratar esta redundancia es mediante el empleo de rachas (runs) en la codificación.

La codificación mediante rachas indica cada vez el próximo elemento de la imagen y cuánto se repite. Si se aplica este método a imágenes en blanco y negro, se puede omitir cuál es el próximo elemento ya que siempre será distinto al anterior (blanco-negro-blanco...), tomando la convención de empezar siempre, por ejemplo, por blanco.

REDUNDANCIA VISUAL

La imagen percibida por el ojo no se corresponde exactamente con la imagen física. Por ejemplo, se ha estudiado que el ojo humano no es capaz de distinguir entre dos niveles de gris parecidos cuando hay involucrados niveles de contraste elevados. Así, desde el punto de vista de nuestra percepción, cierta información puede considerarse menos importante y por tanto redundante.

Idealmente, los métodos de eliminación de redundancia visual modifican la imagen de manera visualmente imperceptible con el objeto de obtener una representación que permita mayor compresión. Esta modificación de la imagen supone una pérdida de información irreversible respecto de la imagen original, y por ello estos métodos se conocen como *compresores con pérdida* o con error (ver Figura 3). Esta pérdida constituye la diferencia fundamental con los métodos precedentes, ya que en aquéllos no se eliminaba información alguna, simplemente se reducía la cantidad de datos necesarios para representarla. El ejemplo más característico de algoritmo de compresión visual es el definido por el estándar JPEG.

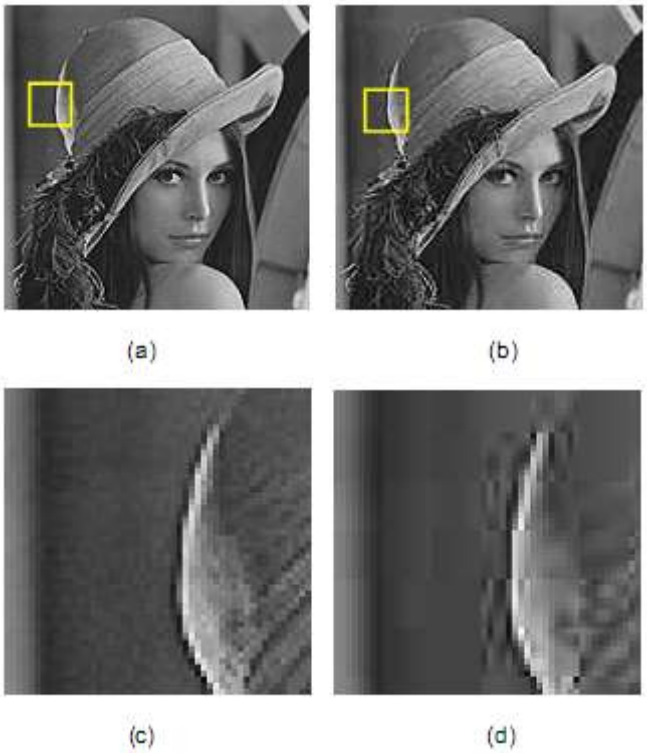


Figura 3.- Efecto de la compresión JPEG, donde se aprecian los artefactos que introduce la compresión con pérdida. (a) Imagen de Lena sin comprimir ocupa 30Kb; (b) comprimiendo con el algoritmo JPEG el tamaño se reduce a 3Kb. (c) detalle del sombrero sin comprimir; (d) detalle del sombrero tras comprimir.

FORMATOS COMERCIALES DE REPRESENTACIÓN

Existen multitud de formatos de ficheros de imágenes de tipo mapa de bits. Se puede hablar de ficheros tipo BMP, TIFF, GIF, JFIF, PGM... Cada uno ofrece ciertas ventajas que otros formatos pueden no contemplar. La Tabla 1 recoge las principales características de algunos de estos formatos.

Formato	Color Real	Paleta	Grises	Bitonal	Compresión	Origen	Multi-Imagen
Bitmap	SI	SI	SI	SI	Run-Length	Windows	NO
TIFF	SI	SI	SI	SI	JPG, LZW, Runs, CCITT4, CCITT3, PackBits	Estándar	SI
JFIF	SI	NO	SI	NO	JPEG	Estándar	NO
JPG2000	SI	NO	SI	NO	JPEG 2000	Estándar	NO
PCX	NO	SI	NO	NO	Propia	Windows	NO
PGM	NO	NO	SI	NO	NO	Unix	NO
GIF	NO	SI	SI	SI	LZW	Estándar	SI

Tabla 1.- Diferentes formatos para ficheros gráficos y características principales.

CONCLUSIONES AL CAPÍTULO

Tras el estudio de este capítulo se comprende la importancia de la adecuada elección de los dispositivos de captura, su configuración y el formato de representación en un sistema informático. Se ha visto que esta elección debe depender del uso que se vaya a dar a la información capturada.

BIBLIOGRAFÍA DEL CAPÍTULO

[VeMo03], [GW93], [Esc01] y [Bax94].

2.1.3 PROCESAMIENTO DE IMÁGENES

En este capítulo se tratan las operaciones y transformaciones que se aplican sobre las imágenes digitales en una etapa de procesamiento previa a las de segmentación y reconocimiento. Su objeto es mejorar o destacar algún elemento de las imágenes, de manera que sea posible realizar las etapas posteriores de segmentación y clasificación.

Todas las operaciones que se van a describir en este capítulo se pueden explicar desde la perspectiva ofrecida por la *teoría de filtros*. Un filtro puede verse como un mecanismo de cambio o transformación de una señal de entrada a la que se le aplica una función, conocida como *función de transferencia*, para obtener una señal de salida. En este contexto se entiende por señal una función de una o varias variables independientes. Los sonidos y las imágenes son ejemplos típicos de señales.

Figura 4.- Esquema de funcionamiento de un filtro.

OPERACIONES BÁSICAS ENTRE PÍXELES

Las operaciones directas sobre píxeles se pueden clasificar en operaciones *aritmético-lógicas* y *operaciones geométricas*.

OPERACIONES ARITMÉTICO-LÓGICAS

Estas operaciones son, con diferencia, las más usadas a cualquier nivel en un sistema de tratamiento de imágenes, ya que son las que se utilizan para leer y dar valores a los píxeles de las imágenes. Las operaciones básicas son:

- **Conjunción.-** Operación lógica AND entre los bits de dos imágenes. Se usa para borrar píxeles en una imagen.
- **Disyunción.-** Operación lógica OR entre los bits de dos imágenes. Se usa para añadir píxeles a una imagen.
- **Negación.-** Inversión de los bits que forman una imagen. Se usa para obtener el negativo de una imagen.
- **Suma.-** Suma de los valores de los píxeles de dos imágenes.
- **Resta.-** Resta de los valores de los píxeles de dos imágenes.
- **Multiplicación.-** Multiplicación de los valores de los píxeles de una imagen por los de otra. Se usa para añadir textura a una imagen.
- **División.-** División de los valores de los píxeles de una imagen entre los de otra.

Se ha visto que en imágenes en niveles de gris se suele utilizar el valor 255 para representar el blanco y el 0 para el negro. Así, la operación de conjunción entre negro y blanco da como resultado negro. Cuando se realiza operaciones aritméticas se debe tener la precaución de verificar que el resultado R de una operación cae dentro del dominio de valores permitidos. En la Figura 5 se pueden apreciar los resultados de diferentes operaciones sobre las imágenes en niveles de gris A y B. Sobre imágenes en color los resultados serían similares.

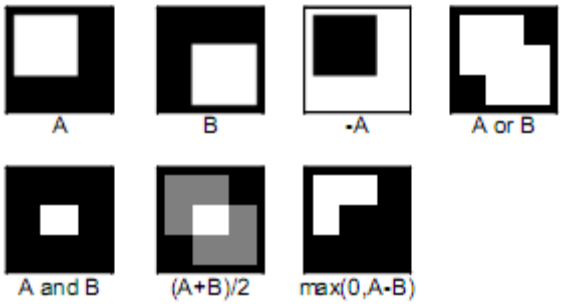


Figura 5.- Ejemplos de operaciones aritméticas y lógicas. Los píxeles a negro corresponden a bits a 0, los blancos a bits a 255.

OPERACIONES GEOMÉTRICAS

Si se expresa los puntos en coordenadas homogéneas, todas las transformaciones se pueden tratar mediante multiplicación de matrices. Las operaciones geométricas más usuales son:

- **Traslación.-** Movimiento de los píxeles de una imagen según un vector de movimiento. La siguiente transformación muestra el resultado de trasladar el punto (x,y) según el vector (d_x, d_y), obteniendo el punto (x', y').

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- **Escalado.-** Cambio del tamaño de una imagen. La siguiente transformación muestra el resultado de escalar el punto (x, y) en un factor (s_x,s_y), obteniendo el punto (x', y').

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- **Rotación.-** Giro de los píxeles de una imagen en torno al origen de coordenadas. La siguiente transformación muestra el resultado de rotar el punto (x, y) un ángulo θ, obteniendo el punto (x', y').

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Las operaciones geométricas matriciales se pueden agrupar multiplicando las matrices. De esta forma, por ejemplo, es posible tener una única matriz que realice un desplazamiento, un giro, otro desplazamiento, y un reescalado en un solo paso. Al realizar esta composición de operaciones se debe recordar que el producto de matrices no cumple la propiedad conmutativa.

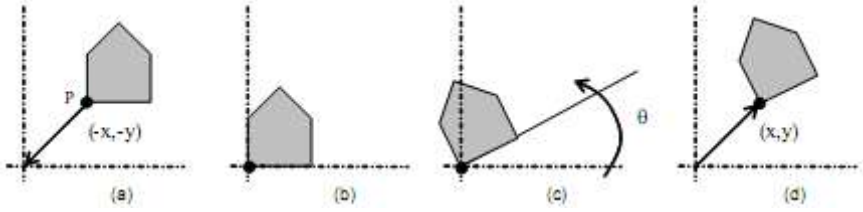


Figura 6.- Ejemplo de rotación. (a) Imagen original que se desea rotar en torno al punto P de coordenadas (x,y); (b) resultado de la primera traslación; (c) resultado del giro; (d) resultado final después de la última traslación.

OPERACIONES SOBRE EL HISTOGRAMA

Se conoce como histograma de los niveles de cuantificación de la imagen, o simplemente histograma de la imagen, a un diagrama de barras en el que cada barra tiene una altura proporcional al número de píxeles que hay para un nivel de cuantificación determinado. Habitualmente, en el eje de abscisas se disponen los diferentes niveles de cuantificación de valores que pueden tomar los píxeles de tal imagen, mientras el eje de ordenadas refleja el número de píxeles que habrá para cada nivel de cuantificación.

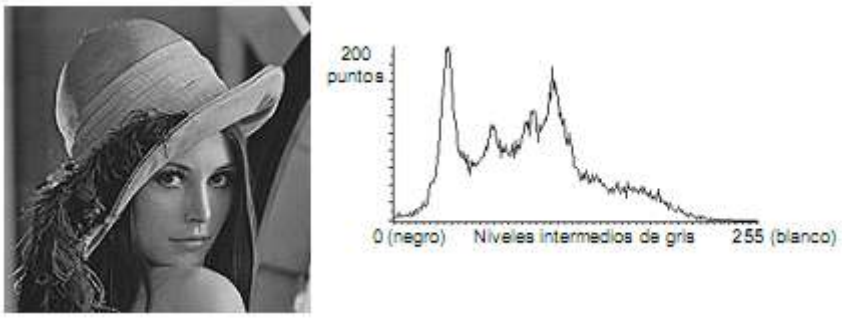


Figura 7.- Imagen en niveles de gris de Lena y su correspondiente histograma.

El histograma de una imagen en niveles de gris proporciona información sobre el número de píxeles que hay para cada nivel de intensidad. En imágenes en color RGB se usan 3 histogramas, uno por cada componente de color.

AUMENTO Y REDUCCIÓN DE CONTRASTE

Las modificaciones del histograma se pueden visualizar eficazmente mediante las *funciones de transferencia del histograma*. Estas funciones corresponden a aplicaciones, pues para cada punto del dominio solo tiene un valor imagen. Estas aplicaciones están acotadas entre 0 y 1 tanto en la abscisa, que se hace corresponder con la entrada IE del filtro, como en la ordenada, que se corresponde con la salida IS del filtro.

En la Figura 8 se presentan tres ejemplos de funciones de transferencia: la *función lineal*, la *función cuadrado* y la *función raíz cuadrada*. La función de transferencia lineal (a) no introduce modificación alguna sobre el histograma, al coincidir exactamente los niveles de intensidad de la entrada y de la salida. La función cuadrado produce un oscurecimiento general de la imagen (en la figura b) se aprecia que el rango entre 0 y 0’5 se hace corresponder con el rango entre 0 y 0’25 que es más oscuro). Por último, la función raíz cuadrada (c) produce un aclarado general de la imagen.

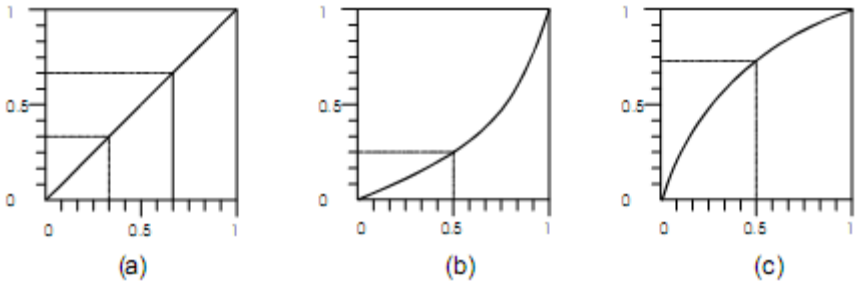


Figura 8.- De izquierda a derecha las funciones lineal, cuadrado y raíz cuadrada.

Una función de transferencia que aclare los niveles claros y oscurezca los niveles oscuros, conseguirá sobre el conjunto de la imagen un efecto visual de aumento de contraste. Una función tal se puede obtener componiendo una función de transferencia del histograma que hasta el valor de 0’5 se comporte como la función cuadrado y que en adelante se comporte como la función raíz. En la Figura 9 se ha representado esta función de transferencia. La función (b) de la misma figura produce el efecto contrario, esto es una disminución del contraste.

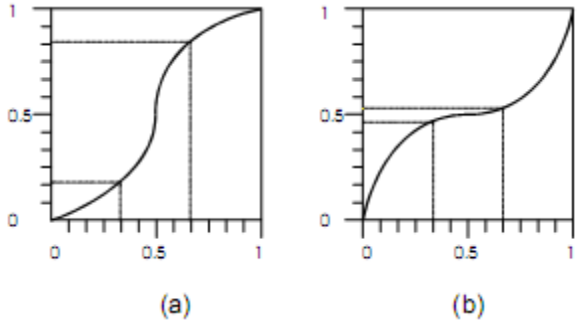


Figura 9.- Funciones de transferencia para aumento y reducción de contraste.

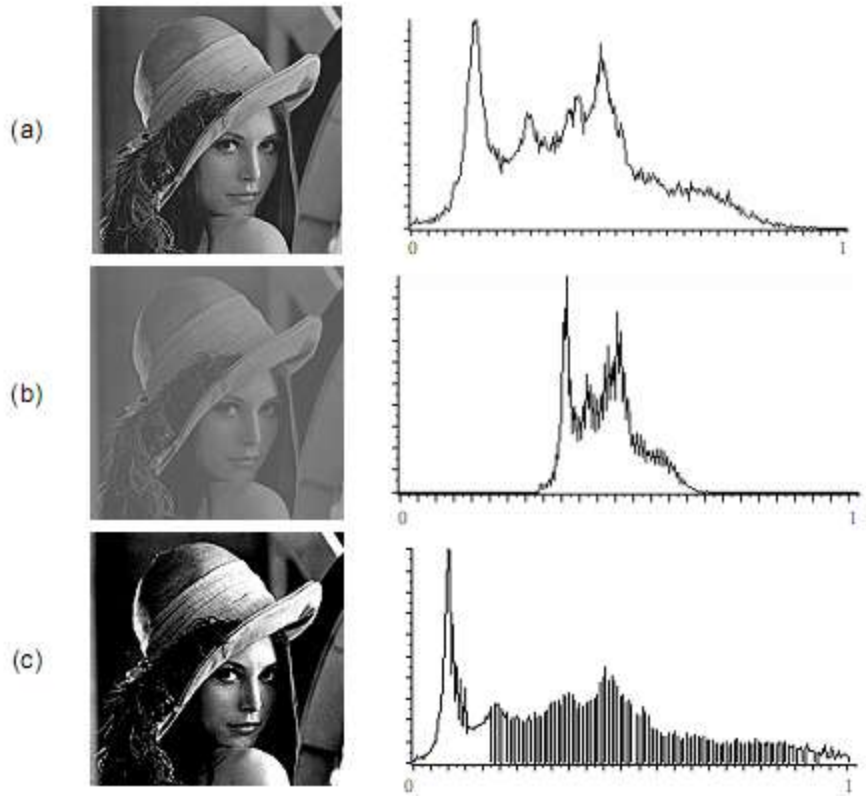


Figura 10.- Transformaciones del histograma sobre la imagen de Lena: (a) imagen original con su correspondiente histograma; (b) resultado de una operación de disminución de contraste; (c) aumento de contraste.

ECUALIZADO DEL HISTOGRAMA

El proceso de ecualizado tiene por objetivo obtener un nuevo histograma, a partir del histograma original, con una distribución uniforme de los diferentes niveles de intensidad.

Al transformar cualquier distribución continua en una distribución uniforme se está maximizando la cantidad de información que contiene. Y aunque ya se ha dicho que en el caso discreto es imposible aumentar la cantidad de información,

el ecualizado del histograma puede mejorar la calidad visual de imágenes parcialmente saturadas. Este efecto se debe a que se cambian los valores de intensidad de las zonas saturadas, en las que originalmente existen objetos que no se distinguen adecuadamente al inspeccionar visualmente la imagen.

Para exponer el ecualizado del histograma se modifica ligeramente la representación del histograma para asimilarla a la de una función de densidad de probabilidad. Para ello se precisa normalizar el histograma, de manera que los niveles de intensidad pasen a ser una variable aleatoria R que varíe entre 0 y 1, y el área del histograma normalizado sea igual a la unidad. Para conseguir esto, se normaliza el número de intensidades a valores entre 0 y 1 y se divide cada elemento del histograma por el número de píxeles de la imagen (para que su suma sea 1).

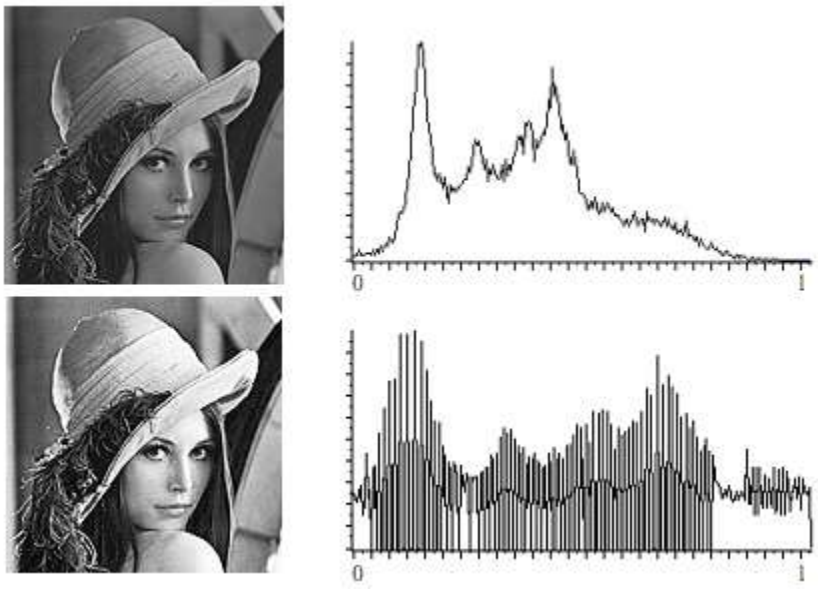


Figura 11.- Ecualizado del histograma sobre la imagen de Lena: (a) imagen original con su correspondiente histograma; (b) ecualizado del histograma.

FILTRADO ESPACIAL

Los filtros espaciales son filtros que se realizan directamente sobre la imagen y por tanto en el dominio del espacio. Aunque hay diferentes tipos de filtros espaciales, los más usados son los *filtros espaciales de convolución*.

FILTROS DE SUAVIZADO

El filtrado de suavizado espacial se basa en el promediado de los píxeles adyacentes al píxel que se evalúa. Quizás el filtro de suavizado más simple que se puede diseñar se corresponde con una matriz de 3x3 con todos los elementos a 1.

El resultado de la convolución de cada píxel se deberá dividir por 9 para asegurar el obtener valores dentro del rango de la paleta. En la figura adjunta se puede apreciar el resultado de la aplicación de este filtro.

Las siguientes matrices de convolución definen otros filtros de suavizado:

$$h = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad h = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Otro filtro de suavizado es el *filtro de la mediana*. Éste se basa en sustituir el valor de un píxel por el de la mediana del conjunto formado por el mismo y sus ocho vecinos.

El *filtro del bicho raro* es otro ejemplo de filtro suavizado. Consiste en comparar la intensidad de un píxel con la de sus 8 vecinos. Si la diferencia es superior a cierto umbral U (que debe elegirse previamente), se sustituye tal píxel por el valor promedio de los píxeles vecinos, en otro caso se mantiene su valor de intensidad.

Debe observarse que tanto el filtro de la mediana, como el filtro del "bicho raro" son filtros no lineales, y por tanto no se pueden obtener mediante una operación de convolución.

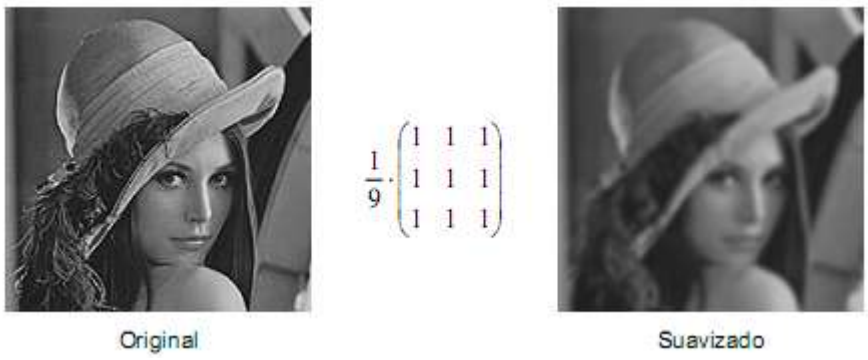


Figura 12.- Aplicación de un filtrado espacial de suavizado.

FILTROS DE OBTENCIÓN DE CONTORNOS

El cálculo de la derivada direccional de una función permite conocer cómo se producen los cambios en una dirección determinada. Tales cambios suelen corresponder a los contornos de los objetos presentes en las imágenes.

Partiendo de que el operador gradiente se define como:

$$\nabla(I(x,y)) = \frac{\partial I}{\partial x}\vec{u}_x + \frac{\partial I}{\partial y}\vec{u}_y$$

Se definen los filtrados de convolución G_x, y G_y:

$$G_x = \frac{\partial I}{\partial x} = I(x,y) * h_1(x,y)$$

$$G_y = \frac{\partial I}{\partial y} = I(x,y) * h_2(x,y)$$

$$h_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \qquad h_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

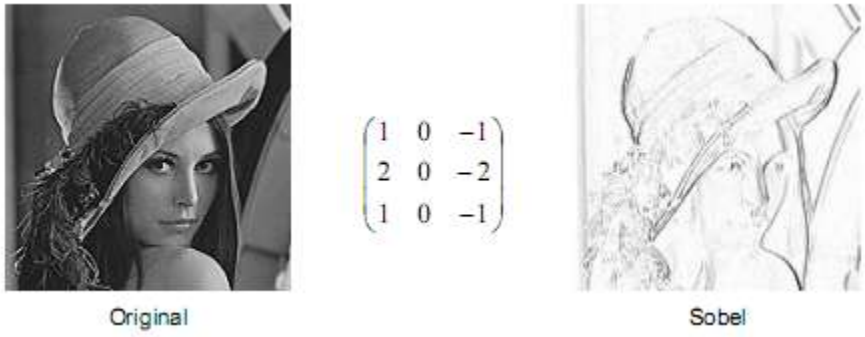
Por ejemplo la matriz h1 proporciona un filtrado en el que un cambio de brillo entre dos píxeles adyacentes en horizontal produce un valor distinto de cero.

En particular los cambios de claro a oscuro se marcan con un valor positivo y los cambios de oscuro a claro con un valor negativo. Por otro lado, cuando dos píxeles adyacentes tienen el mismo valor la convolución con h1 en ese punto devuelve cero.

$$h_1 = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad h_2 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Estas matrices se conocen como *ventanas de Sobel*, que fue quien las propuso. Mediante ellas se calcula el gradiente en las direcciones horizontal y vertical. En la Figura 13 se ve cómo el resultado de aplicar h1 sobre la imagen de Lena produce una imagen en la que aparecen los contornos horizontales de la figura de la imagen original. Ese resultado se obtiene utilizando un factor de división de 4 y presentando el valor absoluto de la convolución, utilizando niveles de gris en escala desde 0 como blanco hasta 255 como negro.

Una alternativa muy común al uso de valor absoluto para evitar los valores fuera de rango consiste en el uso de un factor de suma que se aplica tras la convolución y la división. Por ejemplo, en el caso del filtro de Sobel un factor de división de 8 y un factor de suma de 128 evitarían los valores fuera de rango.



$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

Figura 13.- Filtrado de Sobel en la dirección x en valor absoluto.

Otros sencillos filtros espaciales de localización de contornos que se pueden encontrar en la bibliografía son los de Roberts y los de Prewitt.

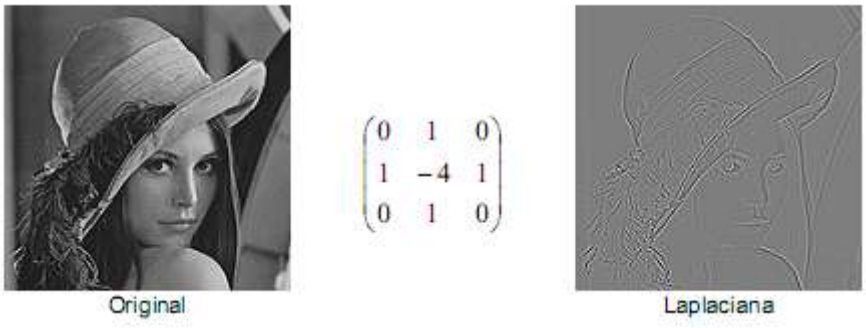
$$\text{Robert}: \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \text{ y } \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \qquad \text{Prewitt}: \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ y } \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

FILTRO DE LA LAPLACIANA

El operador laplaciano de una función bidimensional I(x,y) es el escalar:

$$\Delta(I(x,y)) = \nabla(\nabla(I(x,y))) = \frac{\partial^2 I}{\partial x^2}\vec{u}_x + \frac{\partial^2 I}{\partial y^2}\vec{u}_y$$

Este operador, que se basa en la segunda derivada, se hace cero cuando la primera derivada se hace máximo, es decir cuando aparece un cambio de signo en la primera derivada. Su cálculo parte del de la primera derivada.



$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Figura 14.- Laplaciana de la imagen de Lena. Para su presentación a 255 niveles de gris se ha usado un factor de división de 8 y un factor de suma de 128.

OPERACIONES EN EL DOMINIO DE LA FRECUENCIA

Se ha visto que una imagen digital es una representación que se refiere directamente a la intensidad luminosa de puntos de un espacio, por eso se dice que una imagen digital es una representación en el dominio del espacio. Existen otras representaciones, que contienen la misma información, pero que no están en el dominio del espacio. Es el caso de las representaciones en el dominio de la frecuencia.

Las representaciones en el dominio de la frecuencia, detallan con cuánta frecuencia se repiten ciertos patrones en una imagen, y con ello consiguen representar la información de tal imagen. Esta representación puede ser especialmente útil, ya que teniendo la frecuencia de repetición de tales patrones se pueden detectar y alterar directamente elementos presentes en las imágenes como el ruido, los contornos o las texturas.

Para realizar una transformación al dominio frecuencial es necesario el uso de transformadas:

De forma general, una transformada de una imagen f(x,y) es una función F(u,v) que nos permite representar f como combinación lineal de ciertas funciones base, de modo que:

$$f(x,y) = c\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)a_{u,v}(x,y) \qquad \text{T. Inversa (ec. de síntesis)}$$

siendo c una constante.

Nos interesan, en particular, las transformadas ortogonales (caracterizadas por funciones de base ortogonal) en las que la transformación inversa se obtiene mediante:

$$F(u,v) = C\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)a^{*}_{u,v}(x,y) \qquad \text{T. directa (ec. de análisis)}$$

siendo C una constante.

En general, los parámetros u y v están asociados a frecuencias.

TRANSFORMADA DISCRETA DE FOURIER (DFT)

Utiliza como funciones base imágenes sinusoidales complejas:

$$a_{u,v}(x,y) = e^{ju\frac{2\pi}{M}x + jv\frac{2\pi}{N}y} = \cos\left(u\frac{2\pi}{M}x + v\frac{2\pi}{N}y\right) + j\sin\left(u\frac{2\pi}{M}x + v\frac{2\pi}{N}y\right)$$

Las expresiones de la DFT y su inversa son:

$$f(x,y)=\frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{ju\frac{2\pi}{M}x+jv\frac{2\pi}{N}y}$$

$$F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-ju\frac{2\pi}{M}x-jv\frac{2\pi}{N}y}$$

Observe que, en principio, puede calcularse F(u,v) para cualesquiera valores enteros de u y v.

En todo caso, la ecuación de síntesis sólo precisa los valores de u entre 0 y M-1 y de v entre 0 y N-1.

La extensión de F para el resto de valores enteros es periódica en u y en v con períodos M y N, respectivamente.

VISUALIZACIÓN DE LA DFT

Suele representarse el módulo (fundamentalmente) y la fase. También suele representarse con (0,0) en el centro de la imagen, de modo que las componentes de baja frecuencia quedan centradas y las altas frecuencias en los extremos de la imagen.

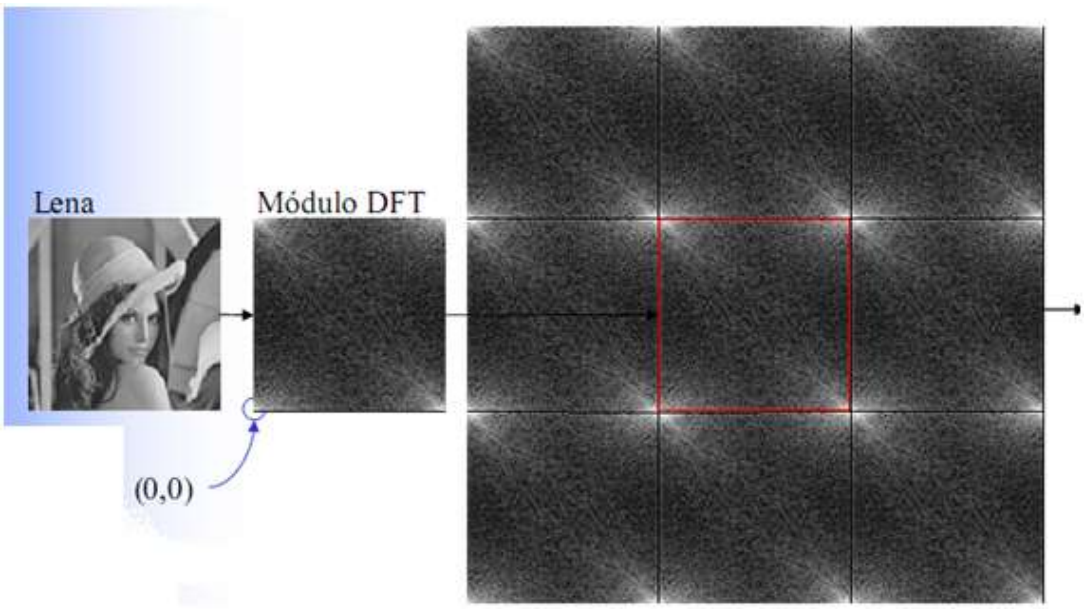


Figura 15.- Representación de la DFT de la imagen de Lena.

PROPIEDADES DE LA DFT

Las propiedades de la DFT son extensiones directas de las propiedades de la TF de señales unidimensionales, entre ellas cabe destacar:

- Su **modulo es simétrico** y su **fase antisimétrica**.
- **Separabilidad**: la DFT puede obtenerse en dos fases, DFT de todas las columnas y DFT de todas las filas del resultado.
- El **modulo** de la DFT es **invariante** frente a **desplazamientos circulares**.
- La DFT posee la propiedad de **convolución** con lo cual se le pueden aplicar **filtros lineales**.

FILTRADO FRECUENCIAL

Una vez conocida la formulación de la transformada de Fourier para imágenes en niveles de gris ya es posible transformar una imagen del dominio del espacio al dominio de la frecuencia. Como ya se ha visto, una vez en el dominio de la frecuencia, es sencillo realizar filtrados que eliminen elementos que aparezcan con cierto periodo.

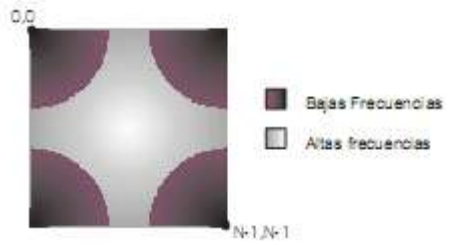


Figura 16.- Situación de los valores correspondientes a las altas y a las bajas frecuencias sobre la matriz de coeficientes de la transformada discreta de Fourier bidimensional.

FILTRO PASO BAJO IDEAL

Los filtros paso bajo son filtros que eliminan las frecuencias altas, dejando "pasar" las bajas frecuencias. Para realizar un filtrado de este tipo basta con poner a cero los módulos de los coeficientes de Fourier relativos a las altas frecuencias, dejando sin modificar los relativos a las bajas frecuencias. Es importante elegir un valor de corte adecuado a partir del cual se considera que una frecuencia es alta o baja.

FILTRO PASO ALTO IDEAL

Los filtros paso alto son filtros que eliminan las bajas frecuencias, permaneciendo las altas frecuencias. Para realizar un filtrado paso alto hay que poner a cero los módulos de los coeficientes de Fourier relativos a las bajas frecuencias, dejando sin modificar los relativos a las altas frecuencias. De nuevo

hay que elegir un valor de corte adecuado a partir del cual se considera que una frecuencia es alta o baja.

FILTROS PASO BANDA IDEAL

Los filtros paso banda son filtros en los que permanece inalterado un rango (o banda) de frecuencias determinado y son eliminados los coeficientes correspondientes al resto de frecuencias. Así, los filtros paso alto y paso bajo constituyen dos casos límites del filtro paso banda.

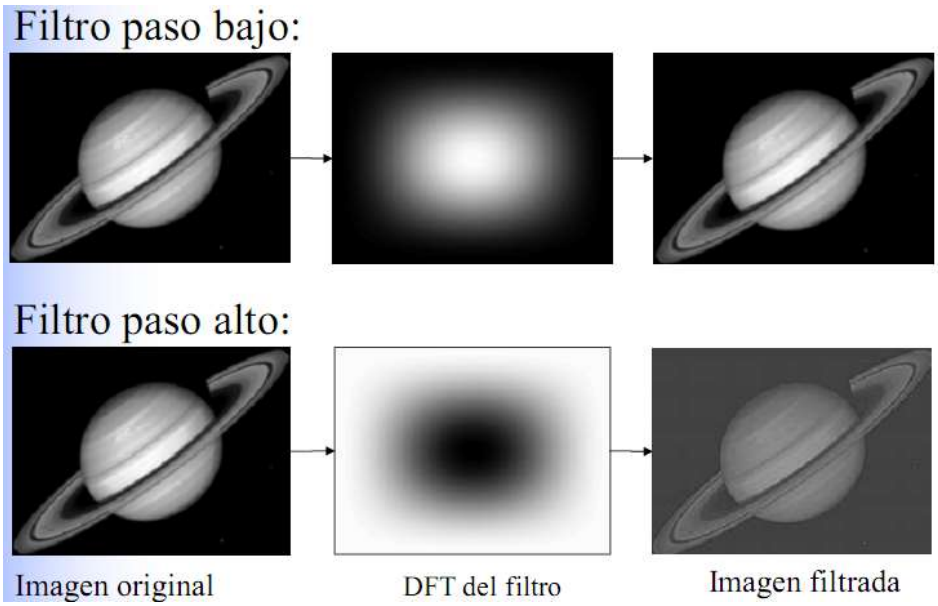


Figura 17.- Ejemplos de filtrado paso bajo y paso alto.

OTROS OPERADORES EN EL DOMINIO DE LA FRECUENCIA

Existen multitud de operadores que también operan en el dominio de la frecuencia y que no se estudian en este capítulo por motivos de espacio. En general su operativa es similar a la estudiada para la Transformada de Fourier. Sin embargo, cada uno aporta sus propias ventajas e inconvenientes.

No se debe terminar esta sección sin citar las transformadas de onditas (wavelets) que suponen una generalización al concepto de transformada de Fourier en el que la base del espacio vectorial no son las funciones senos y cosenos sino cualquier tipo de función dentro de una amplia familia (Haar, Gaussiana, Mexican Hat, etc...). Estos operadores están obteniendo gran aplicación debido a la particularización que supone elegir adecuadamente la función de la base para cada problema concreto y a la reducción del coste computacional de los algoritmos asociados a su cálculo respecto a la transformada de Fourier. Así, por ejemplo, el estándar de compresión de imágenes JPEG2000 usa compresión por onditas.

OPERACIONES MORFOLÓGICAS

Clásicamente la morfología ha sido una parte de la biología que estudia la forma de los animales y de las plantas. De la misma forma, la *morfología matemática* consiste en un conjunto de técnicas matemáticas que permiten tratar problemas que involucran formas en una imagen. En este apartado se tratará detalladamente la morfología sobre imágenes bitonales, para luego presentar los operadores básicos sobre imágenes en niveles de gris.

El lenguaje que se usa en las técnicas morfológicas proviene de la teoría de conjuntos. Cada conjunto representa la forma de los objetos en una imagen:

- Imágenes binarias: cada conjunto de todos los píxeles negros/blancos de una imagen binaria, es una descripción completa de la imagen: Si

  Blanco = 1

  Negro = 0

  En una imagen binaria, cada elemento de A o B es un punto de coordenadas (x,y) en el plano bidimensional (de la imagen).

- Imágenes monocromáticas: una imagen de niveles de gris puede ser representada como conjuntos cuyas componentes se encuentran en un espacio 3D.

  En este caso, 2 componentes de cada elemento de un conjunto se refiere a las coordenadas del píxel, y la tercera componente está relacionada con la intensidad.

ELEMENTO ESTRUCTURANTE

Los operadores morfológicos analizan la estructura geométrica de una imagen usando como sonda un patrón de ajuste que se denomina elemento estructurante (E.E.). Desplazando el EE sobre la imagen, el operador analiza típicamente su posición en relación al primer plano y al fondo de la misma.

Figura 18.- Posibles posiciones del EE.

El EE puede tener cualquier tamaño y forma (círculo, cuadrado, etc.). Su centro se sitúa en cada píxel de la imagen original, aplicando la operación morfológica sobre los puntos situados bajo el E.E.

DILATACION

La salida de la dilatación es el conjunto de puntos barridos por el centro del EE mientras algún punto del EE coincide con alguno de la imagen. Alternativamente (véase figura 19), puede interpretarse la dilatación como el resultado de reemplazar cada píxel blanco de la imagen original por una réplica del elemento estructurante.

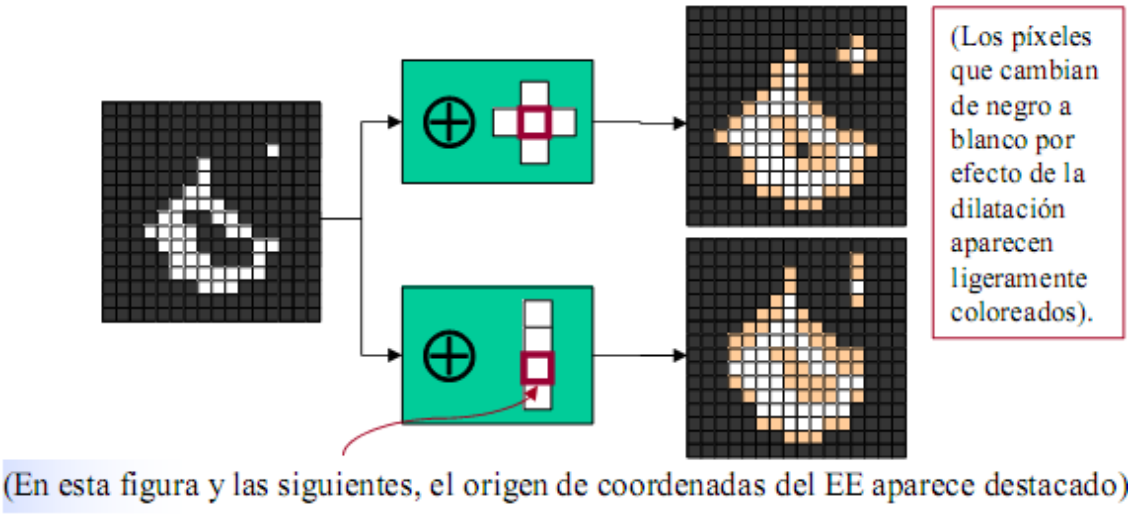


Figura 19.- Ejemplo de dilatación.

Rellena entrantes en los que no quepa el EE (pequeños agujeros y bahías):

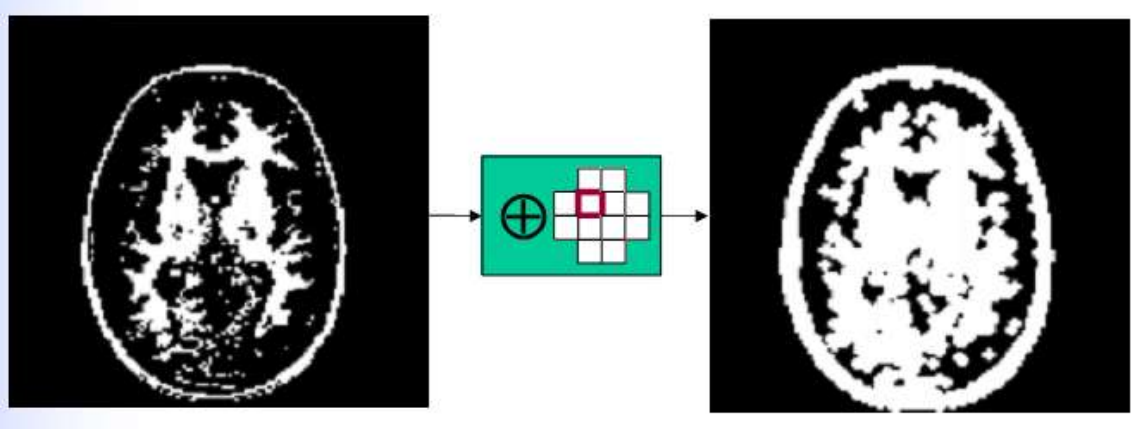


Figura 20.- Ejemplo de dilatación en la imagen de un cerebro.

EROSION

La salida de la erosión es el conjunto de puntos barridos por el centro del EE mientras se cumpla que todos los puntos del EE estaban contenidos en la imagen.

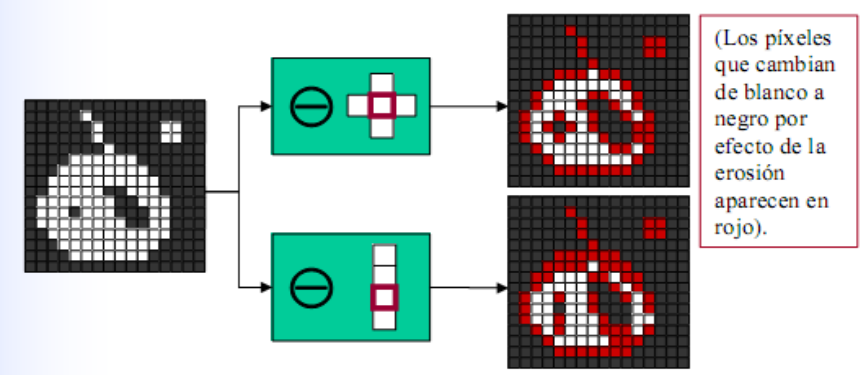


Figura 21.- Ejemplo de erosión.

Elimina grupos de píxeles donde el EE no cabe:

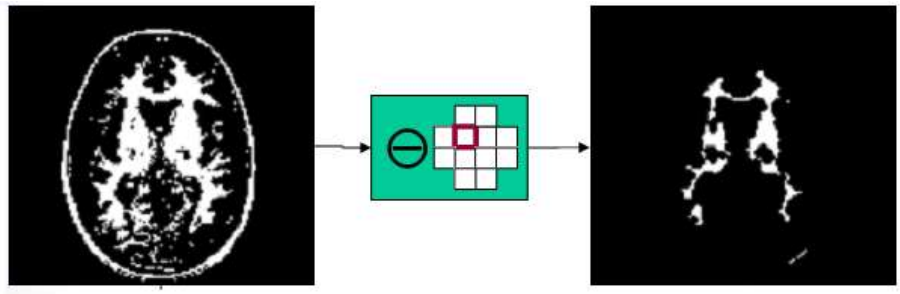


Figura 22.- Ejemplo de erosión en la imagen de un cerebro.

Una de las aplicaciones más típicas de la erosión es la eliminación de detalles irrelevantes (desde el punto de vista de tamaño):

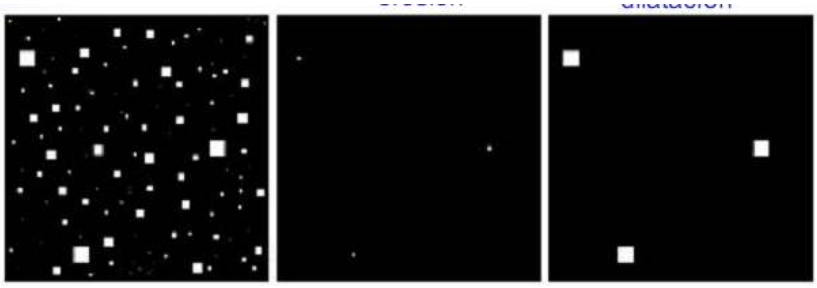


Figura 23.- En esta figura se desea obtener solo los objetos de mayor tamaño, para ello erosionamos con un EE de tamaño algo menor que el objeto que queremos extraer. El resultado son los objetos en cuestión pero de menor tamaño. Para que recuperen el tamaño original se aplica una dilatación con el mismo EE.

FILTROS MORFOLOGICOS

APERTURA

Es la composición de un operador de erosión y otro de dilatación con el mismo elemento estructurante.

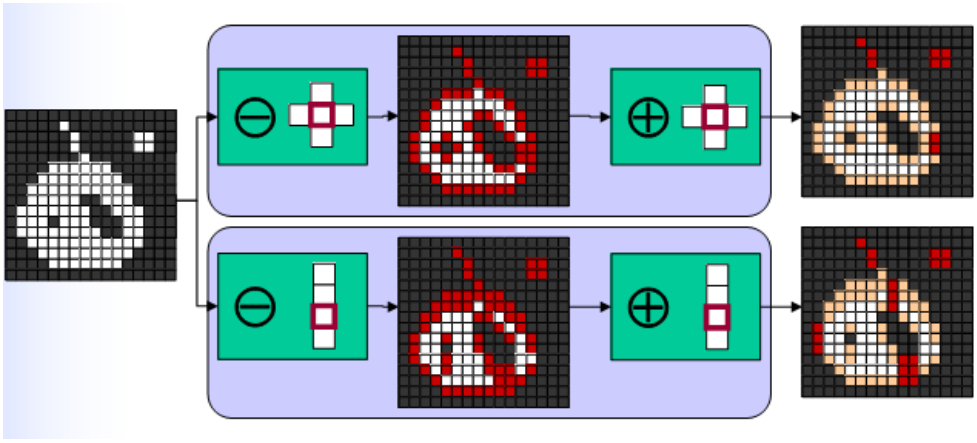


Figura 24.- Ejemplo de apertura.

Algunos efectos:

- Alisa contornos (redondea las esquinas donde no quepa el EE).
- Eliminar las protuberancias donde no quepa el EE.
- Separa objetos en puntos estrechos.

CIERRE

Es la composición de un operador de dilatación seguido de otro de erosión con el mismo elemento estructurante

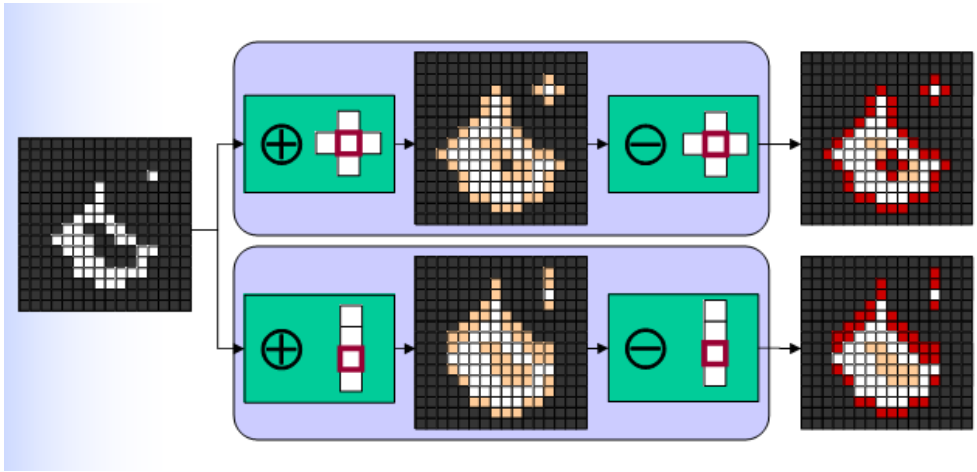


Figura 25.- Ejemplo de cierre.

Algunos efectos:

- Tiende a alisar porciones del contorno, fusiona estrechas grietas, y rellena vacíos en el contorno.
- Rellena agujeros pequeños.
- Elimina entrantes pequeños.
- Conecta objetos vecinos.

MORFOLOGÍA DE GRISES

La morfología de grises (más de 2 niveles considera una dimensión más que las imágenes binarias, la altura representada por la intensidad de cada píxel).

Existen dos tipos de elementos estructurantes:

- *morfología plana:* cuando se utiliza un elemento estructurante plano (mismo valor en todos sus elementos: "1" ó "0").
- *morfología no plana:* cuando los valores del EE son distintos.

MORFOLOGÍA PLANA

Dilatación: si los valores del EE son positivos, la imagen resultante tiende a ser más brillante que la imagen de entrada, los detalles oscuros son reducidos o eliminados, dependiendo de sus valores y forma con respecto al EE.

Erosión: si EE es positivo, la imagen resultante tiende a ser oscura, se reduce el efecto de los detalles brillantes que son más pequeños que el EE, siendo el grado de reducción dependiente de los niveles de gris que rodean al detalle y los valores y forma del EE.

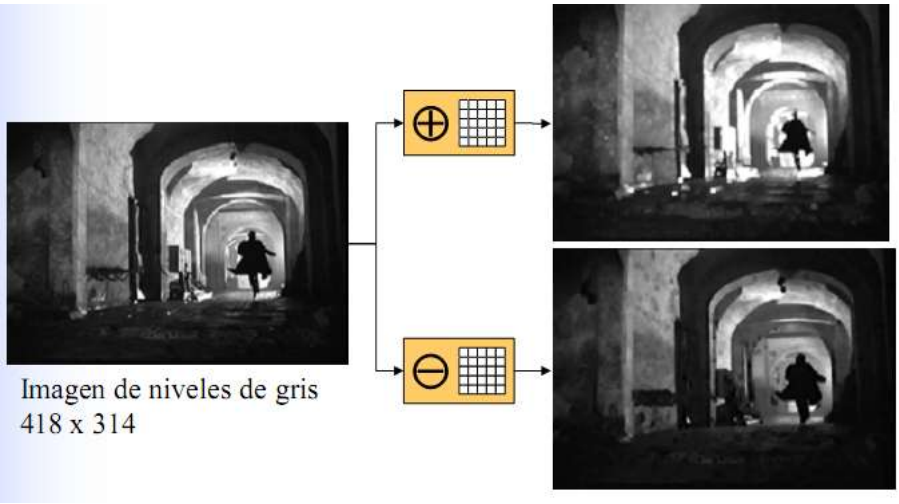


Figura 26.- Ejemplo de dilatación y erosión con morfología plana.

Apertura y cierre: la apertura y cierre simplifican las imágenes (equivalen a un filtro paso bajo no lineal), tanto en imágenes binarias como de grises. Se eliminan picos y valles más estrechos que el elemento estructurante. Se definen del mismo modo que en el caso binario:

- Apertura: erosión seguida de dilatación.
- Cierre: dilatación seguida de erosión

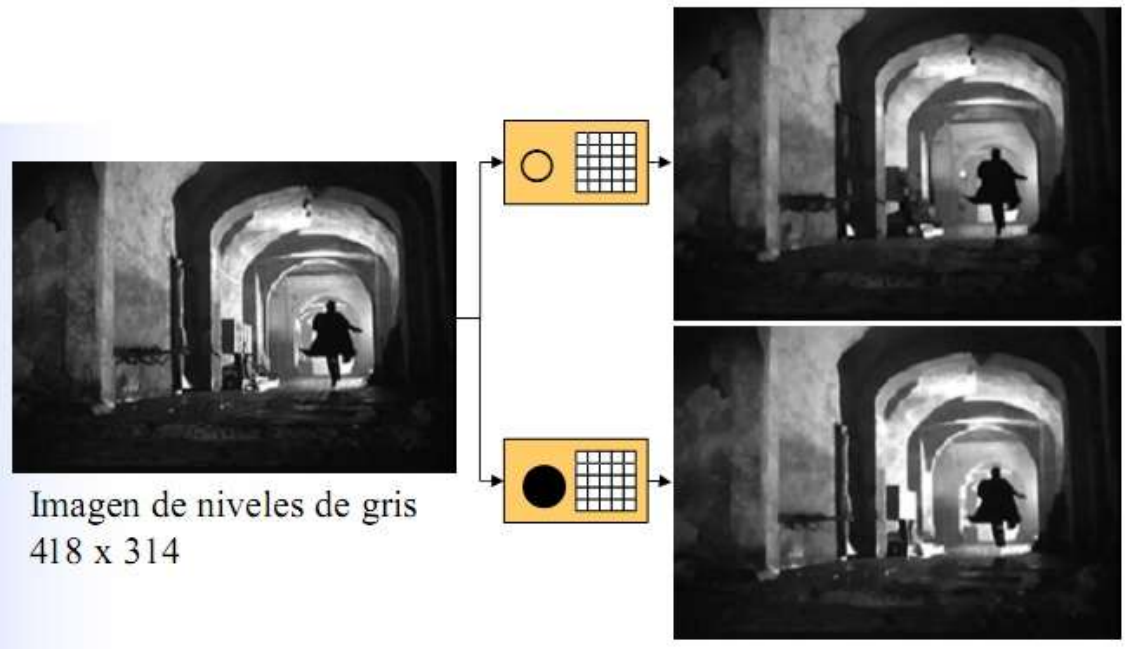


Figura 27.- Se observa como la apertura reduce los detalles brillantes, mientras que el cierre reduce los detalles pequeños y oscuros.

CONCLUSIONES AL CAPÍTULO

En este capítulo se han estudiado diferentes operaciones que usadas adecuadamente facilitan las etapas posteriores de análisis de las imágenes.

Sin embargo, debe recordarse que realizando estas u otras manipulaciones nunca se gana información, sólo se promociona o se descarta información ya existente en la imagen. Por ello siempre que la calidad de una imagen sea pobre, resultando insuficiente para el uso al que se destina, debe pensarse en la posibilidad de variar las condiciones de captura.

BIBLIOGRAFÍA DEL CAPÍTULO

[VeMo03], [GW93], [JKS95] y [Esc01].

2.1.4 SEGMENTACIÓN

La segmentación es un proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características (como por ejemplo el brillo o el color) con el fin de facilitar un posterior análisis o reconocimiento automático. Localizar la cara de una persona dentro de la imagen de una fotografía o encontrar los límites de una palabra dentro de una imagen de un texto, constituyen ejemplos de problemas de segmentación.

A pesar de que existen diferentes enfoques para realizar la segmentación en la práctica se demuestra que la segmentación no tiene reglas estrictas a seguir, y dependiendo del problema en cuestión, puede ser necesario idear técnicas a medida.

Figura 28.- Ejemplo de segmentación en el que se extraen de la imagen llaves y monedas.

CONCEPTOS BÁSICOS SOBRE SEGMENTACIÓN

La segmentación debe verse como un proceso en el que a partir de una imagen se produce otra en la que cada píxel tiene asociada una etiqueta distintiva del objeto al que pertenece. Así, una vez segmentada una imagen, se podría formar una lista de objetos consistentes en las agrupaciones de los píxeles que tengan la misma etiqueta.

La segmentación termina cuando los objetos extraídos de la imagen se corresponden unívocamente con las distintas regiones disjuntas a localizar en la misma. En este caso se habla de *segmentación completa* de la escena o imagen y en el caso contrario, de *segmentación parcial*. En una escena compleja, el resultado de la segmentación podría ser un conjunto de regiones homogéneas superpuestas y en este caso, la imagen parcialmente segmentada deberá ser sometida después a un tratamiento posterior con el fin de conseguir una segmentación completa.

En general, el proceso de la segmentación suele resultar complejo debido, por un lado, a que no se tiene una información adecuada de los objetos a extraer y, por otro, a que en la escena a segmentar aparece normalmente ruido. Es por esto que el uso de conocimiento sobre el tipo de imagen a segmentar o alguna otra información de alto nivel puede resultar muy útil para conseguir la segmentación de la imagen.

Algunos ejemplos típicos de procesos de segmentación son: tratar de separar los caracteres que forman una palabra dentro de una imagen de un texto, detectar ciertos tipos de células en imágenes médicas, extraer los vehículos que aparecen en una imagen de una carretera.

Los diferentes objetos que aparecen en una imagen pueden ser localizados atendiendo a aspectos como: sus contornos o su textura. Podemos diferenciar entre 3 grupos de técnicas: técnicas basadas en umbralización, basadas en detección de los contornos de los objetos y técnicas basadas en propiedades locales de las regiones.

LA TEXTURA

Intuitivamente la textura de un objeto dentro de una imagen es el conjunto de formas que se aprecia sobre su superficie y que lo dota de cierto grado de regularidad. Una definición clásica de textura es la siguiente: “uno o más patrones locales que se repiten de manera periódica”.

Existen dos enfoques para definir una textura: uno descendente (“top-down”) y otro ascendente (“bottom-up”). El enfoque descendente se basa en la existencia de un elemento básico de textura, llamado téxel, y en una regla de formación. Esta regla define cómo y dónde se sitúan estos elementos básicos. Este enfoque funciona bien cuando la textura es bastante regular, por ejemplo en la imagen de una pared de ladrillos. Por otro lado, el enfoque ascendente se basa en que la textura es una propiedad que se puede derivar de estadísticos (como la media y la varianza) de pequeños grupos de píxeles. Este enfoque funciona bien para texturas donde resulta difícil ver los componentes individuales, por ejemplo la textura de la hierba o el cuarzo. No obstante, la línea divisoria entre los dos enfoques no es clara.

EL CONTORNO

El contorno de un objeto en una imagen digital corresponde al mínimo conjunto de píxeles que separa ese objeto del fondo o background de la imagen. Normalmente estos contornos se corresponden con los puntos donde se producen discontinuidades en los valores de píxeles adyacentes (cambios en el matiz o el brillo) o con los puntos donde cambia un patrón que se repite (cambios de textura).

SEGMENTACIÓN BASADA EN LA UMBRALIZACIÓN

La umbralización es un proceso que permite convertir una imagen de niveles de gris o de color en una imagen binaria, de tal forma que los objetos de interés se etiqueten con un valor distinto al de los píxeles del fondo. En adelante sólo se hablará de imágenes en niveles de gris, aunque la extensión a color es inmediata si sólo se usa una de las componentes RGB o alguna mezcla de las tres.

La umbralización es una técnica de segmentación rápida, que tiene un coste computacional bajo y que puede ser realizada en tiempo real durante la captura de la imagen usando un computador personal de propósito general.

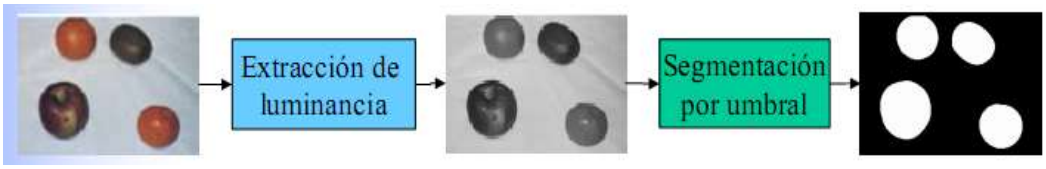


Figura 29.- Extracción de piezas de fruta mediante segmentación por umbral.

Por otro lado el histograma de una imagen no tiene en cuenta la información espacial sino solamente la distribución de grises en la imagen. Por ello, dos imágenes muy diferentes pueden tener el mismo histograma. Esto hace que, los métodos de segmentación basados en la umbralización, como único medio de segmentación, resulten limitados en muchos problemas reales. Aunque sí se usan con frecuencia como complemento de otros métodos.

UMBRALIZACIÓN FIJA

El caso más sencillo, conocido como *umbralización fija*, se puede usar en aquellas imágenes en las que existe suficiente contraste entre los diferentes objetos que se desea separar. Consiste en establecer un valor fijo sobre el histograma que marque el umbral de separación. Para obtener dicho umbral se debe disponer de información sobre los niveles de intensidad de los objetos a segmentar y del fondo de la imagen. De esta forma, la imagen binaria resultante B(i, j) se define a partir de la imagen digital original I(i, j) en función de un valor U que corresponde al umbral de separación seleccionado según la siguiente fórmula.

$$B(i,j) = \begin{cases} 1, \text{si } I(i,j) \geq U \\ 0, \text{si } I(\text{i},\text{j}) < U \end{cases}$$

La elección de un valor de umbral correcto resulta decisiva para llevar a cabo la segmentación de una imagen de manera satisfactoria. La obtención del umbral

suele basarse en el histograma de la imagen. Cuando en el histograma se aprecian uno o más lóbulos, éstos suelen corresponder con una o varias zonas de la imagen, que comparten niveles de intensidad similares. Estos objetos pueden ser directamente los objetos a segmentar o corresponder a partes homogéneas de objetos más complejos. Lógicamente, la transición de un lóbulo a otro se corresponde con un mínimo del histograma, correspondiendo estos mínimos a los puntos que fijan el valor umbral. La búsqueda de dichos mínimos (basada por ejemplo en el cálculo de derivadas) se encuentra dificultada por la naturaleza ruidosa del histograma. Para atenuar este problema puede aplicarse un filtro paso bajo sobre el histograma de la imagen.

La Figura 30 ilustra el proceso descrito. En ella se distinguen una zona de tierra y otra de mar en una foto de satélite (a). El histograma (b) presenta dos lóbulos que tras ser suavizados (c) usando un filtro espacial de paso bajo, muestra un mínimo de separación en el valor 42. Eligiendo dicho mínimo como umbral de separación entre tierra y mar, se obtiene el resultado de la figura (d).

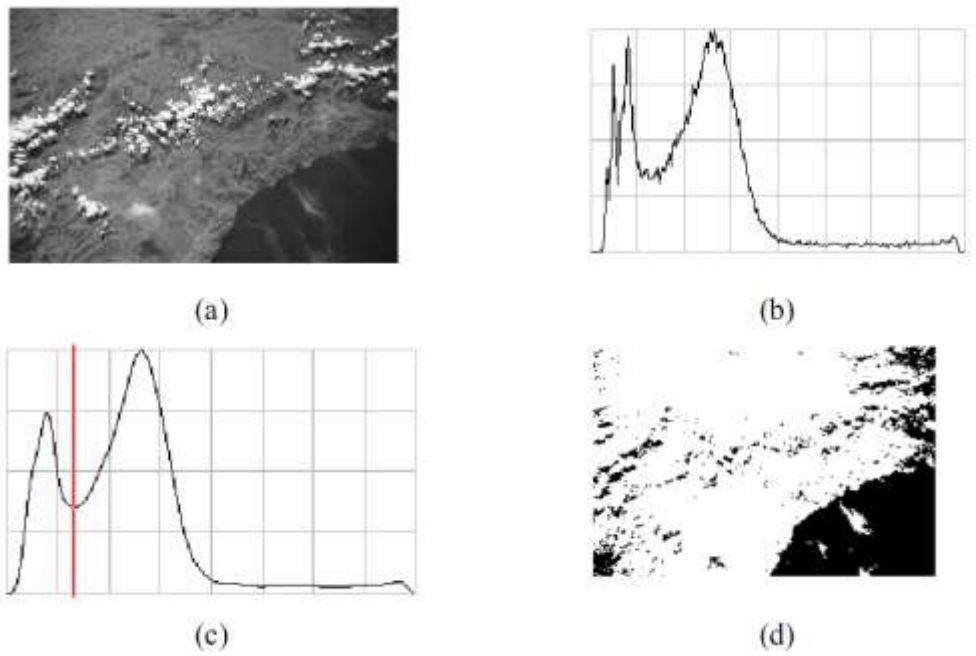


Figura 30.- La elección adecuada del umbral de binarización permite separar la tierra del mar en esta imagen de satélite, aunque la sombra de las nubes sobre la tierra y las nubes sobre el mar producen ciertos errores. Sería preciso realizar algún pos-proceso para obtener una segmentación completa.

UMBRALIZACIÓN GENERALIZADA.

En general, la obtención de un único valor de umbral fijo no es útil en imágenes complejas. Por ejemplo, sobre imágenes de documentos con fondos complejos o sobrecenas con iluminación no uniforme, el estudio del histograma de la imagen

puede revelar la inexistencia de un único valor umbral que permita separar los objetos del fondo o background. Esto lleva a considerar otros tipos de umbralización que resultan de generalizar la idea de umbral. A continuación, se definen la *umbralización de banda*, la *multiumbralización*, la *semiumbralización* y la *umbralización adaptativa*.

UMBRALIZACIÓN DE BANDA

La umbralización de banda permite segmentar una imagen en la que los objetos (regiones de píxeles) contienen niveles de gris dentro de un rango de valores y el fondo tiene píxeles con valores en otro rango disjunto. Así,

$$B(i,j) = \begin{cases} 1 & \text{si } I(i,j) \in R \\ 0 & \text{en otro caso} \end{cases}$$

donde R representa un rango de valores correspondientes a niveles de gris que definen a los elementos a extraer de la imagen digital. Multiumbralización La multiumbralización, como su nombre indica, consiste en la elección de múltiples valores de umbral dentro del proceso, permitiendo separar a diferentes objetos dentro de una escena cuyos niveles de gris difieran. El resultado no será ahora una imagen binaria sino que los diferentes objetos (regiones) tendrán etiquetas diferentes:

$$\begin{aligned} I_S(i,j) &= 1, && \text{si } I(i,j) \in R_1 \\ &= 2, && \text{si } I(i,j) \in R_2 \\ &= 3, && \text{si } I(i,j) \in R_3 \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ &= n, && \text{si } I(i,j) \in R_n \\ &= 0, && \text{en otro caso} \end{aligned}$$

donde I(i, j) es la imagen original, IS(i, j) es la imagen segmentada y R1, R2, …,Rn representan los *n* diferentes rangos de niveles de gris usados para umbralizar.

SEMIUMBRALIZACIÓN

La semiumbralización persigue obtener una imagen resultado en niveles de gris, y para ello pone a cero el fondo de la imagen conservando los niveles de gris de los objetos a segmentar que aparecen en la imagen inicial. Su formulación es:

$$I_S(i,j) = \begin{cases} I(i,j) & \text{si } I(i,j) \geq U \\ 0 & \text{en otro caso} \end{cases}$$

UMBRALIZACIÓN ADAPTATIVA

En las técnicas anteriores, los rangos de umbralización se consideran fijos con independencia de las características locales de la imagen considerada. En muchas imágenes, donde la iluminación no es uniforme, puede ocurrir que píxeles del mismo objeto a segmentar tengan niveles de gris muy diferentes. Ello conlleva que no sea posible elegir un único umbral que, sobre toda la imagen, distinga los píxeles de un objeto de los de otro. La umbralización adaptativa o variable permite resolver este problema haciendo que el valor del umbral varíe según una función que depende de las características locales del entorno del punto que se evalúa.

TÉCNICAS BASADAS EN LA DETECCIÓN DE CONTORNOS

La segmentación basada en detección de contornos agrupa un gran número de técnicas que usan la información proporcionada por las fronteras de los objetos que aparecen en una imagen.

Puesto que se desea encontrar los objetos individuales presentes en una imagen, parece lógico que si se encuentran las fronteras de tales objetos con el fondo se podría segmentar los objetos de la escena general.

SEGMENTACIÓN BASADA EN LAS COMPONENTES CONEXAS

"Para todo píxel p de una imagen, el conjunto de los píxeles hasta los que hay un camino desde p se dice que forman su componente conexa. Además se cumple que dos componentes conexas distintas tienen conjuntos de píxeles disjuntos".

Utilizando el concepto de componente conexa se puede plantear el detectar los objetos presentes en una imagen sin más que encontrar las componentes conexas de la misma. Esto ocurre cuando los objetos tienen un color uniforme y distinto del fondo, lo que permite asegurar que los contornos del objeto se corresponden con los bordes de la componente conexa.

En el caso de imágenes en blanco y negro las componentes conexas suelen corresponder a los objetos directamente. Así por ejemplo en un documento escaneado los objetos de color negro suelen ser los caracteres que se desea reconocer, y un etiquetado de componentes conexas basta para encontrarlos (ver

Figura 31). Esto también ocurre en el caso de imágenes a contraluz, o de imágenes umbralizadas.

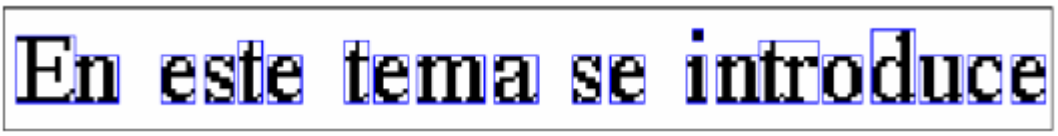


Figura 31.- Ejemplo de segmentación basada en componentes 8-conexas.

En el caso de imágenes en niveles de gris o en color, la elección del conjunto de puntos V que definen la conectividad es fundamental para que el etiquetado de componentes conexas aporte información válida sobre los objetos de una imagen.

De cualquier manera, el estudio de las componentes conexas suele ser un paso fundamental para la segmentación, aunque insuficiente por sí solo. Puede observarse en la Figura 31 que la t y la r de la palabra "introduce" forman la misma componente conexa. Por ello, posteriormente suele aplicarse heurísticas particulares para conseguir una segmentación completa.

DETECCIÓN DE CONTORNOS CON FILTROS DE GRADIENTE

En el caso de que los objetos no tengan un color uniforme, o que este color pueda cambiar dependiendo del objeto y del fondo, es preciso utilizar técnicas que permitan detectar cambios entre los valores de los píxeles de las imágenes, más que fijar de antemano cuáles corresponden a objetos y cuáles no.

En general, la segmentación basada en detección de contornos presenta varios problemas. El más importante quizás consiste en la no aparición de un contorno o frontera que sí exista en la imagen real. Además, junto con los contornos suele aparecer ruido que se deriva de la propia naturaleza de la imagen y esto hace que pueda aparecer un contorno fantasma que no exista en la realidad.

Por otra parte, tras el filtrado y el umbralizado, los contornos suelen aparecer con un grosor apreciable de varios píxeles, mientras que sería deseable que los contornos sólo tuviesen un píxel de grosor. Por último, los contornos obtenidos rara vez aparecen de forma conexa, por lo que en vez de una línea que describe el contorno del objeto de manera continua, se tiene una serie de fragmentos de los contornos de mayor o menor longitud. Para solucionar estos problemas suelen usarse enfoques y algoritmos basados en heurísticas particulares.

Tanto el filtro de la primera derivada (basado en el operador de gradiente) como el de la segunda (basado en el operador de la laplaciana) son muy sensibles al ruido, por ello suele aplicarse previamente un filtro de suavizado para eliminarlo. El filtro del gradiente suele producir contornos gruesos, mientras que

el filtro de la laplaciana, suele producir imágenes con el grosor de los contornos a un píxel. Sin embargo, el filtro de segunda derivada es más sensible al ruido que el de gradiente. Por ello suelen usarse combinados considerándose como contorno aquellos píxeles donde el módulo del gradiente supera un umbral y además se produce cambio de signo en la segunda derivada, que corresponde a un paso por cero. Hay que notar que raramente coincide el valor cero con el valor de un píxel tras la aplicación de la laplaciana; ese cero se produce a resolución subpíxel y sólo es detectable por un cambio de signo en el resultado de la segunda derivada.

Finalmente, para localizar los contornos a partir de la imagen resultado del filtrado suelen aplicarse ciertos algoritmos que procesan los resultados y devuelven los segmentos que corresponden a los posibles contornos. En los siguientes puntos se presentan tres técnicas distintas que hacen esto.

TRANSFORMADA DE HOUGH

La *transformada de Hough* es un método de análisis global que se diseñó para detectar líneas rectas y curvas a partir de las posiciones de n puntos. Una ventaja de esta técnica es la robustez de los resultados de segmentación conseguidos al aplicarla; sin embargo, su coste computacional es elevado.

Figura 32.- Rectas y curvas encontradas en una imagen de un campo de futbol mediante la transformada de Hough.

El algoritmo propuesto por Hough en 1962, conocido como transformada Hough, permite determinar el conjunto de rectas que probablemente forman una nube de puntos. Este algoritmo parte de la consideración de que para cualquier punto (xi,yi), todas las rectas que pasan por él cumplen la ecuación:

$$y_i = a\,x_i + b$$

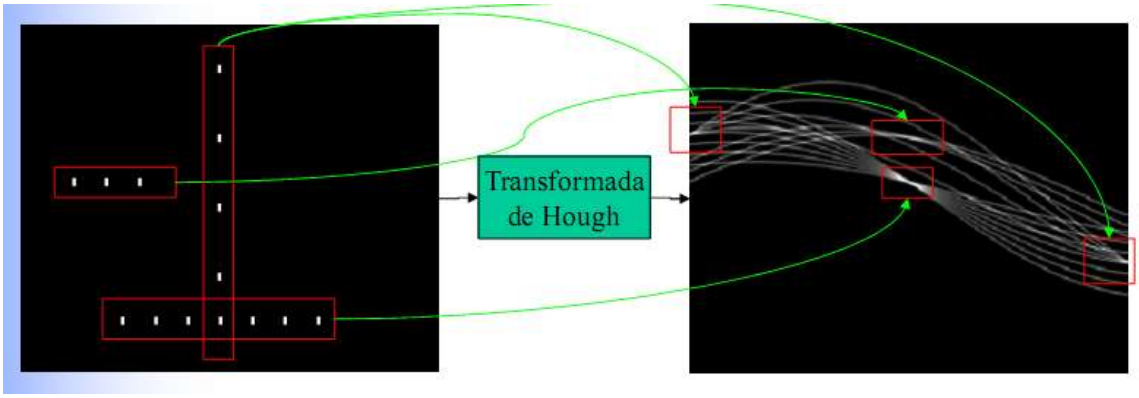


Figura 33.- Trasformada de Hough de puntos alineados. El brillo en la transformada es proporcional al número de puntos que contribuyen.

TÉCNICAS BASADAS CRECIMIENTO DE REGIONES

Las técnicas agrupadas bajo el nombre de *crecimiento de regiones* determinan zonas dentro de una imagen basándose en criterios de similaridad y proximidad entre los píxeles de la misma. En estas técnicas la homogeneidad (o falta de homogeneidad) entre regiones adyacentes es el criterio utilizado para unir (o dividir) regiones de la imagen. Dicha homogeneidad se puede definir a partir de criterios como: el nivel de gris medio, el color, la forma, etc. El resultado de la segmentación es una partición de la imagen en regiones homogéneas.

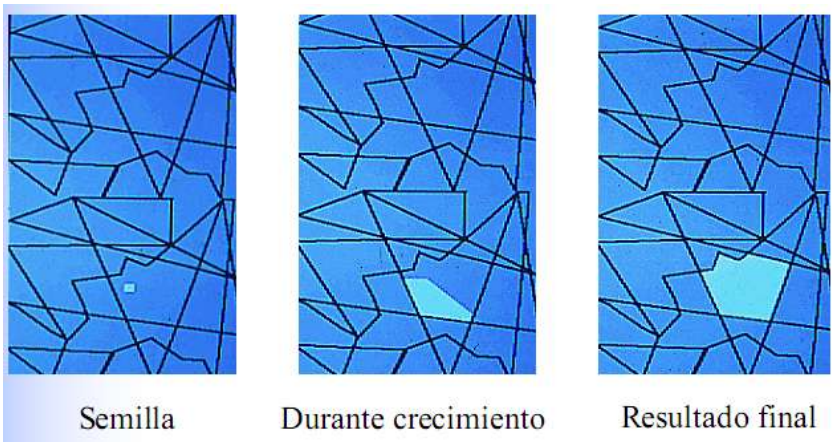


Figura 34.- El crecimiento de regiones parte de un punto o semilla y va creciendo incluyendo dentro de su misma región píxeles vecinos con características (nivel de gris, textura, color) similares.

Las técnicas de segmentación basadas en contornos tratan de encontrar fronteras entre regiones. En general, las técnicas basadas en regiones trabajan mejor en imágenes con ruido, en donde los contornos son difíciles de localizar. La segmentación resultante de una detección de contornos y la basada en crecimiento de regiones, aplicadas a una misma imagen no producen

normalmente el mismo resultado. Es posible combinar los resultados producidos por ambos tipos de segmentaciones.

OTROS ENFOQUES PARA LA SEGMENTACIÓN

Actualmente, la segmentación de imágenes continúa siendo un área activa de investigación. Existen numerosas técnicas de segmentación, aparte de las explicadas hasta ahora, que no pueden ser englobadas estrictamente en ninguno de los tres grupos descritos. Por ello, en los siguientes puntos, se repasa, brevemente y de manera separada, las técnicas basadas en el uso del color, en el análisis de la textura y en el movimiento (obtenido como una sucesión de imágenes consecutivas en el tiempo).

SEGMENTACIÓN BASADA EN EL COLOR

El aumento de resolución radiométrica siempre aporta nueva información que puede facilitar el proceso de segmentación. Sin embargo siempre debe tenerse en cuenta que los requisitos computacionales aumentan considerablemente respecto a las técnicas basadas en imágenes en niveles de gris o bitonales.

Ya se ha estudiado que el color se puede representar como la unión de tres planos, cada uno con la información relativa a la intensidad de cada punto respecto a cada una de las componentes de una base de color (rojo, verde y azul en el modelo RGB). Una técnica común de segmentación en color consiste en separar el proceso en dos fases. En la primera se aplican las técnicas que se han estudiado para niveles de gris a cada uno de los tres planos RGB. En la segunda se integran los resultados de la primera para producir como resultado la segmentación de la imagen en color (véase la figura 35).

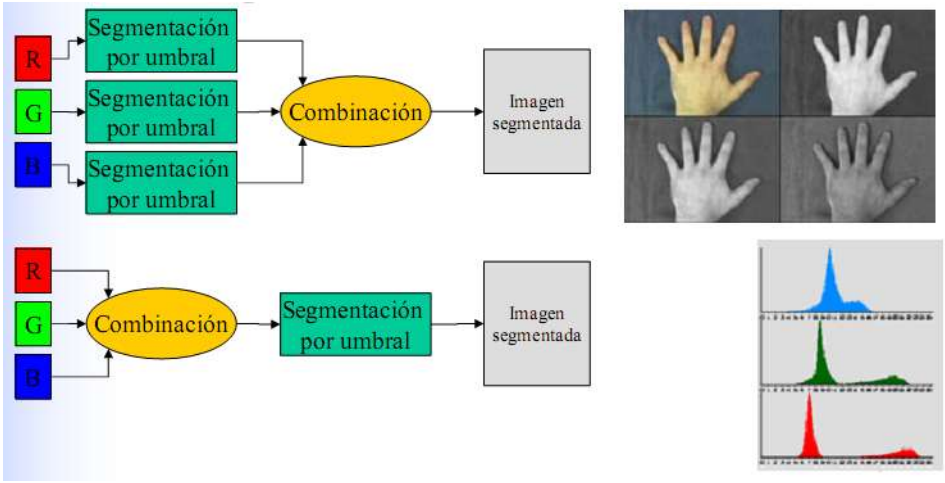


Figura 35.- Se aplican las técnicas de segmentación por umbral a cada canal por separado y posteriormente se combinan para formar la imagen segmentada.

La Figura 36 muestra otro ejemplo de segmentación de imagen en color.

Figura 36.- Ejemplo de segmentación de una imagen en color.

SEGMENTACIÓN BASADA EN LA TEXTURA

En este enfoque se definen *modelos de texturas* para pensar en una imagen no como en una colección de píxeles, sino para tratarla como una función I(x,y). El propósito del modelo es transformar una ventana de una imagen en una colección de valores que constituyen un vector de características. Este vector será un punto de un espacio n-dimensional. La representación corresponderá a la textura si las ventanas tomadas de la misma muestra de textura están "cercanas" en el espacio de características, y si las ventanas de la imagen con diferentes patrones de texturas quedan "alejadas" en el espacio de características considerado. Los modelos de textura se dividen, a grandes rasgos, en tres categorías: basados en *estructuras piramidales*, que tratan de capturar las frecuencias espaciales a distintos niveles de resolución; basados en *campos aleatorios*, que asumen que los valores de un píxel son seleccionados mediante un proceso estocástico bidimensional; y los basados en *métodos estadísticos*, que utilizan matrices de coocurrencias construidas a partir de las imágenes. De estas matrices se extraen una serie de medidas como la media, la varianza, la entropía, la energía y la correlación entre los píxeles.

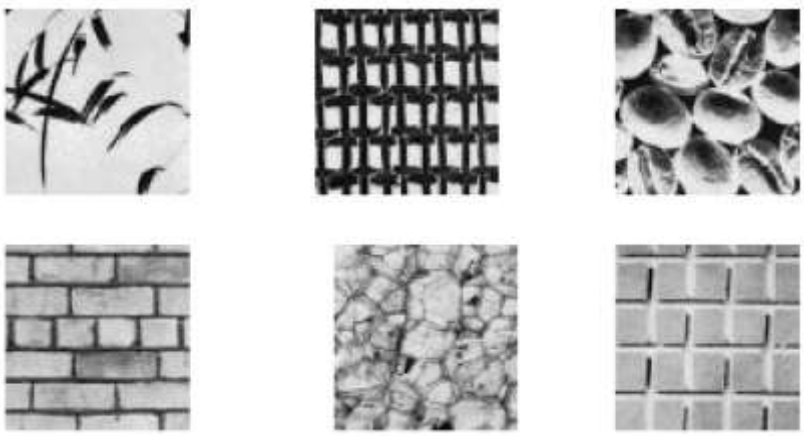


Figura 37.- Diferentes tipos de texturas.

Una vez caracterizada la textura de una imagen hay que aplicar un algoritmo de segmentación que podrá ser supervisado o no supervisado. La diferencia entre ambos enfoques radica en un conocimiento a priori o no de la tarea específica que el algoritmo lleva a cabo (en otras palabras, en el supervisado se conocen de antemano los tipos de texturas presentes y en el no supervisado, no se conocen).

Si se asume que el número de texturas diferentes presentes en la imagen es pequeño y que todas las texturas son distintas unas de otras, entonces es posible describir regiones pequeñas de textura homogénea, extraer vectores de características usando los modelos mencionados, y usar estos vectores como representantes de clases en el espacio de características. Ahora, todos los demás vectores de textura se pueden etiquetar asociándose al representante de clase más cercano. Se pueden usar redes neuronales u otro tipo de algoritmos, para ajustar mejor el sistema al modelo. Se está, en este caso, usando un *método de segmentación supervisado.*

Si sucede que el número de texturas posibles es muy grande y no se pueden realizar suposiciones sobre los tipos de texturas presentes en la imagen, se puede recurrir a usar *métodos de segmentación no supervisados*. Ahora se necesita realizar un análisis estadístico sobre la distribución de vectores de características.

El objetivo es reconocer clústeres o agrupaciones de vectores en la distribución y asignar la misma etiqueta a los componentes de cada uno de ellos.

La siguiente imagen, muestra un ejemplo de imagen texturada y de cómo resultaría su segmentación.

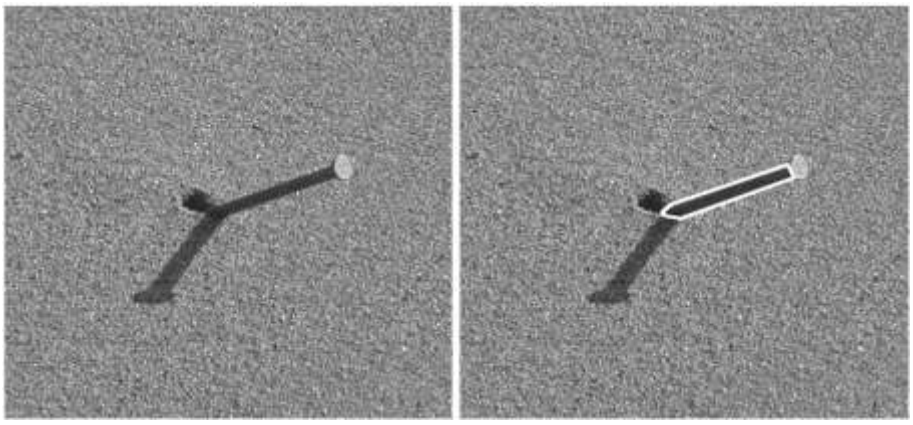


Figura 38.- (a) Imagen inicial, y (b) resultado de su segmentación.

SEGMENTACIÓN BASADA EN EL MOVIMIENTO

El movimiento puede constituir una potente herramienta para la segmentación de objetos animados sobre fondos estáticos. Las técnicas básicas consisten en el

estudio de la imagen resultante de la resta de dos imágenes consecutivas de una secuencia animada. Está técnica se conoce con el nombre de *substracción de fondo*. Los objetos que se desplazan entre estas dos imágenes producen en la imagen resta un conjunto de píxeles con valores distintos a cero. Mientras, los elementos estáticos de la imagen, por no variar, producen cero tras la resta (véase figura 39).

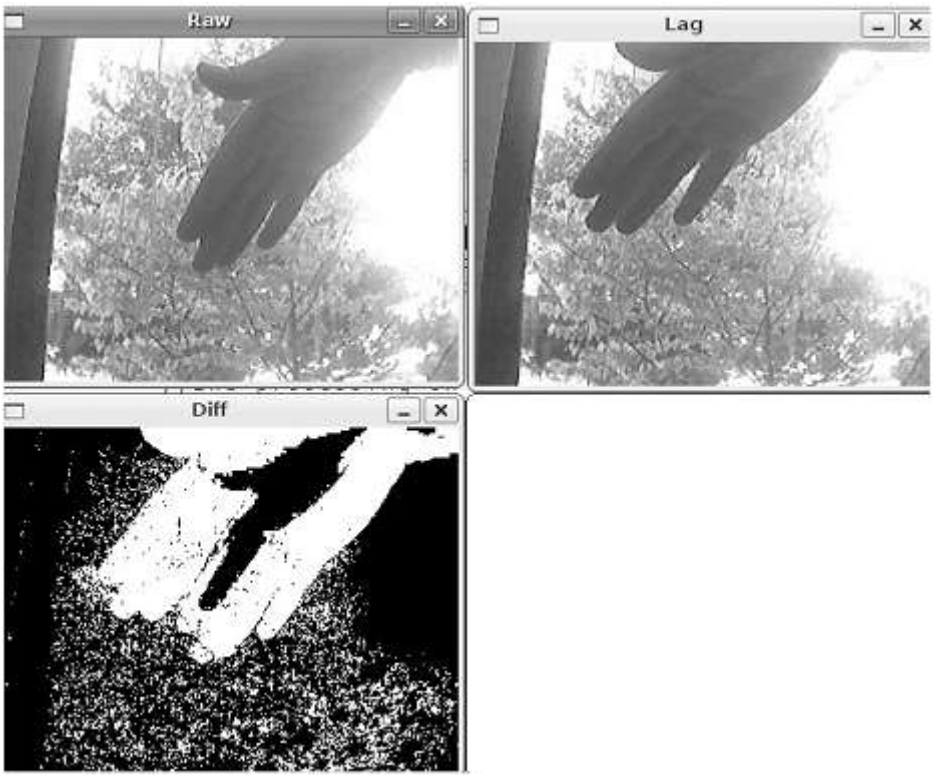


Figura 39.- Entre las imágenes *Raw* y *Lag* se ha producido un movimiento que se ve reflejado en la imagen *Diff* , que es la resta de las dos anteriores.

Así, partiendo de dos imágenes I_t e I_{t+1} de dos instantes consecutivos los objetos tras esta segmentación serían los píxeles a uno en la imagen I_d.

$$d_{t,t+1}(x,y) = \begin{cases} 1 & \text{si } |I_t(x,y) - I_t(x,y)| > U \\ 0 & \text{en otro caso} \end{cases}$$

Siendo U un valor umbral que depende de la variación de la iluminación entre los instantes t y t+1.

BIBLIOGRAFÍA DEL CAPÍTULO

[VeMo03], [GW93], [SHB99], [IMP], [BK08], [AK89], [Ru95], [Tec99] y [Dai].

2.1.5 REPRESENTACIÓN Y DESCRIPCIÓN

Tras la segmentación, las regiones se deben representar de una forma adecuada para su posterior procesado.

La representación de regiones (objetos) es conceptualmente similar a un proceso de compresión, se persigue representar la imagen (posiblemente con pérdidas) de forma sencilla, con pocos parámetros.

Hay dos maneras posibles de descripción de regiones:

- Por su frontera: parámetros de forma.
- Por su interior: parámetros de color, textura, etc.

Los descriptores permiten capturar información esencial sobre un objeto importante para alguna aplicación (de reconocimiento, por ejemplo).

REPRESENTACIÓN

Entre los métodos de representación más importantes podemos encontrar: *códigos de cadenas, aproximaciones poligonales, signaturas, segmentos frontera y esqueletos.*

CÓDIGOS DE CADENAS

Se utilizan para representar una frontera mediante una secuencia conectada de segmentos rectilíneos de longitud y dirección especificadas.

El número de direcciones posibles depende del tipo de vecindad:

Figura 40.- Direcciones posibles en los códigos de cadenas para vecindad a 4 y a 8.

Fijado un punto de inicio, el codificador recorre la curva en la dirección de las agujas del reloj. El remuestreo permite reducir la longitud de la descripción.

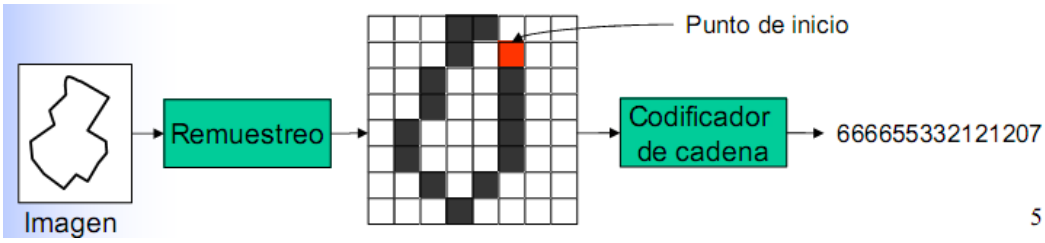


Figura 41.- Ejemplo de codificación de cadenas con vecindad a 8.

Existen códigos invariantes:

- Respecto al punto de inicio: el punto de inicio se elige de modo que minimice el número resultante (*código normalizado*).

  66665533212107 → NORMALIZACIÓN → 076666553321212

- Frente a rotaciones: calcula el número de cambios de dirección (en el sentido contrario a las agujas del reloj). La primera cifra indica el cambio de dirección del último al primer componente de la cadena (*código de primera diferencia*).

  076666553321212 → PRIMERA DIFERENCIA → 677000706077171

APROXIMACIONES POLIGONALES

Persiguen describir una curva mediante un polígono con un número reducido de segmentos rectos.

Para una curva cerrada, la aproximación es exacta cuando el número de segmentos en el polígono es igual al número de puntos en la curva. En la práctica, lo que se persigue es obtener la representación de la frontera del objeto con el menor número de segmentos del polígono.

Idealmente, interesa obtener aproximaciones que minimicen alguna medida de error basada en la distancia de los puntos de la curva a puntos o segmentos de la aproximación poligonal. De este modo, el método de aproximación debe resolver el siguiente problema: dados N puntos, ¿dónde colocarlos para que el error con la frontera original sea mínimo? En general, la resolución de este problema puede ser compleja, y es necesario recurrir a otras alternativas.

POLÍGONO DE PERÍMETRO MÍNIMO

Ajusta la curva por el polígono de menor perímetro que pasa por las celdas que recorre la curva. Para ello, la frontera se acota dentro de una serie de celdas concatenadas. Como resultado tenemos a la frontera confinada entre las paredes de las celdas.

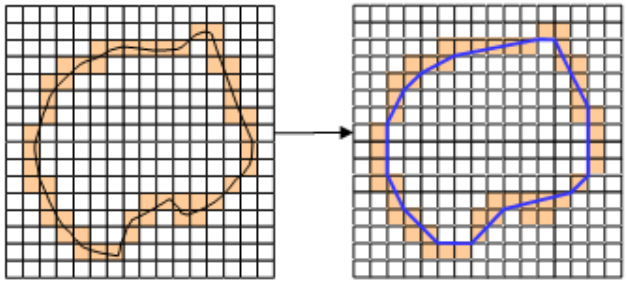


Figura 42.- Ejemplo de aproximación poligonal por perímetro mínimo.

TÉCNICAS DE FUSIÓN

Fusiona pares de puntos en uno solo a lo largo de la frontera hasta que la suma de errores cuadráticos de ajuste supere cierto umbral.

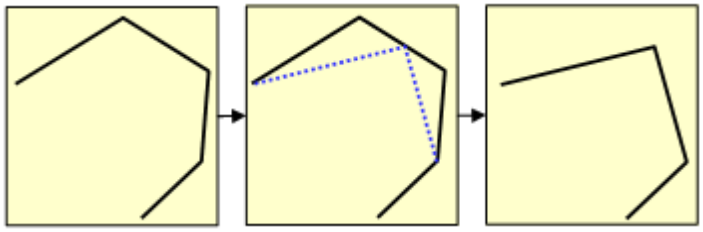


Figura 43.- Ejemplo de aproximación poligonal por técnicas de fusión.

TÉCNICAS DE DIVISIÓN

Divide segmentos en 2 partes de forma sucesiva hasta que se satisfaga un criterio de ajuste.

Por ejemplo, para una frontera cerrada (véase figura 5):

- Puntos de inicio: los 2 puntos más lejanos (a y b).
- Si la distancia máxima de la frontera a la línea supera cierto umbral, divide la línea.

Un ejemplo de umbral puede ser 0.25 veces la distancia del segmento ab.

La división se realiza convirtiendo el punto más lejano (c) en un nuevo vértice.

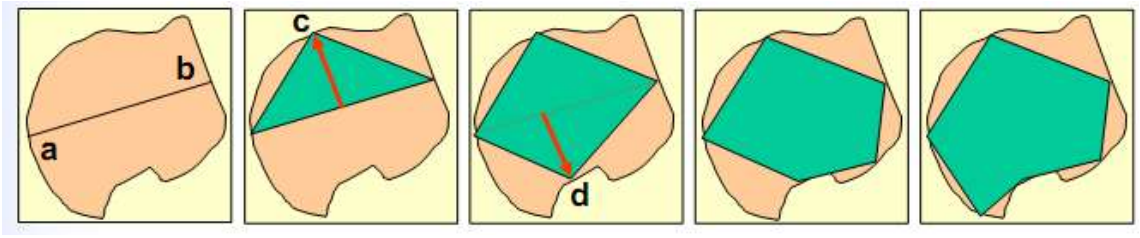


Figura 44.- Ejemplo de técnicas de división.

Para una curva abierta, puede emplearse un procedimiento similar, utilizando como punto de inicio los extremos de la curva. Puede combinarse con estrategias de fusión.

SIGNATURAS

Son representaciones 1-D de la frontera.

Por ejemplo se representa una curva cerrada mediante la función r(θ), distancia del centro geométrico a la frontera en función del ángulo.

Es invariante frente a traslaciones, pero no frente a rotaciones o escalados.

La invarianza frente a rotaciones puede conseguirse eligiendo adecuadamente el punto de inicio, de forma que sea el mismo independientemente de la orientación del objeto:

- El punto más alejado del centro geométrico.
- El punto, en la dirección del eje principal, más alejado del centro geométrico.
- El punto de inicio de un código de cadena de primera diferencia

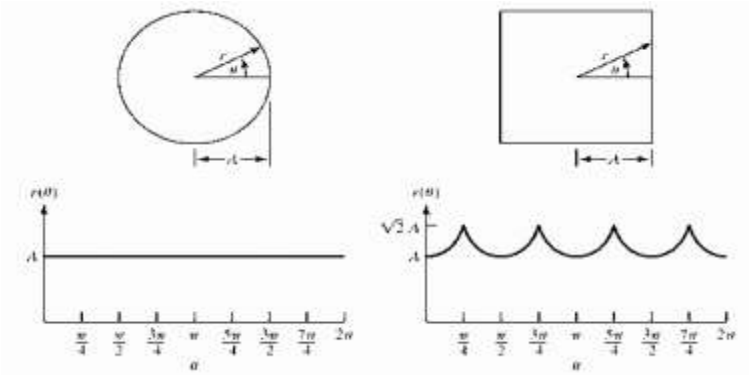


Figura 45.- Método para realizar la descripción por signatura.

SEGMENTOS FRONTERA

Siguen el principio de dividir la frontera en segmentos cuya descripción es más sencilla.

Por ejemplo, se determina la cubierta convexa. Recuérdese que la cubierta convexa H, de un conjunto arbitrario S, es el conjunto convexo más pequeño que contiene al conjunto S. Los puntos de contacto de la cubierta con la frontera determinan los segmentos. Al conjunto diferencia H-S se denomina *Deficiencia Convexa* D del conjunto S. Cada segmento se caracteriza mediante la función de deficiencia convexa.

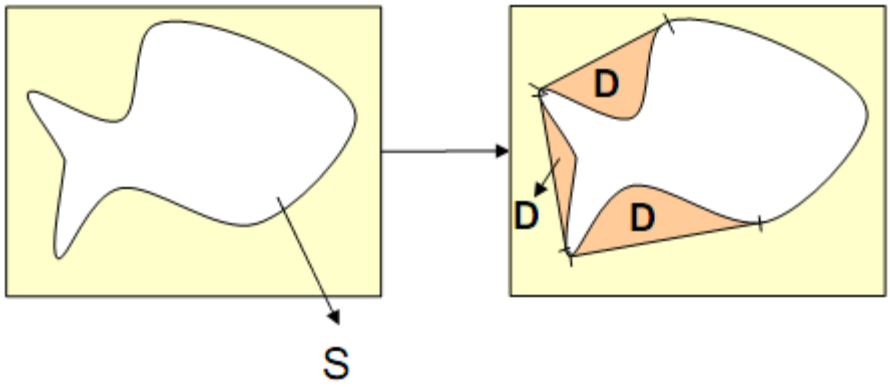


Figura 46.- Determinación de la cubierta convexa de la silueta de un pez.

ESQUELETOS

El esqueleto permite describir una región mediante un grafo. El esqueleto puede obtenerse mediante morfología matemática, aunque estas técnicas no siempre garantizan un esqueleto conectado.

Existen alternativas como el algoritmo MAT (Medial Axis Transformation): sea R una región y B su frontera, para cada punto p en R, encuentra su vecino más próximo en B. Si p tiene más de un vecino más próximo, entonces pertenece al esqueleto.

La carga computacional puede ser alta.

Figura 47.- Proceso de esqueletización de una imagen binaria.

DESCRIPCIÓN

Entre los tipos más importantes de descriptores podemos encontrar los *descriptores frontera y los de región.*

DESCRIPTORES DE FRONTERA

Dentro de los descriptores frontera existen diferentes tipos:

- Descriptores simples, como pueden ser longitud, diámetro y curvatura (tasa de cambio de la pendiente de la frontera, es muy sensible al ruido,

por lo que suele obtenerse a partir de versiones suavizadas o aproximaciones poligonales).

- Descriptores de forma:
  - Número de forma: se denomina así al número que forma el código de cadena de primera diferencia de menor magnitud.
  - Orden del número de forma: número de dígitos en su representación

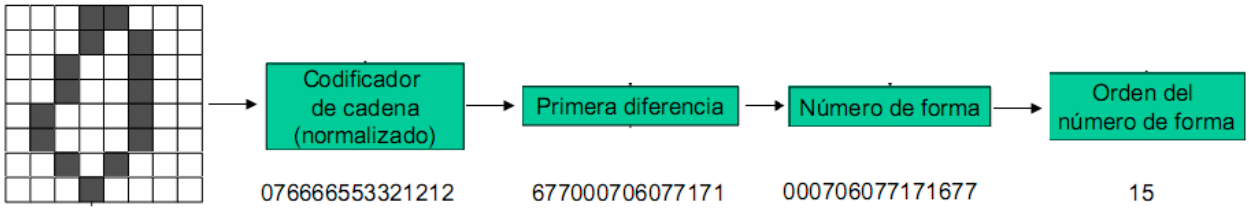


Figura 48.- Ejemplo de un descriptor de forma.

- Descriptores de Fourier: dependiendo del número de coeficientes que se usen se podrán representar fronteras de forma más o menos precisa.
- Momentos estadísticos: pueden obtenerse para cualquier representación 1-D de una imagen. Sean:
  - g(u), la representación 1-D de la imagen.
  - p(v), el histograma de las amplitudes de g (normalizado a suma 1).
  - L, el número de niveles del histograma.

El momento (central) de orden n se define como:

$$\mu_n = \sum_{i=0}^{L-1} (v_i - m)^n p(v_i)$$

Siendo:

$$m = \sum_{i=0}^{L-1} v_i p(v_i)$$

Casos particulares:

- μ2 → dispersión de la curva sobre el valor medio
- μ3 →simetría respecto a la media

DESCRIPTORES DE REGIÓN

Podemos distinguir entre:

- Descriptores simples: perímetro (longitud de su frontera), área (numero de pixeles de la región), compacidad (coeficiente de perimetro^2/área),

mediana y media de los niveles de gris, etc. La compacidad es invariante frente a cambios de escala, frente a un giro, etc.

- Descriptores topológicos: la topología se dedica al estudio de propiedades invariantes frente a deformaciones que no alteran las relaciones de vecindad (esto es, sin rupturas ni fusiones).

  Número de componentes conectadas, C.

  Número de agujeros, H.

  Número de Euler: E = C-H.

  Para la extracción de las componentes conectadas se usan las técnicas morfológicas.

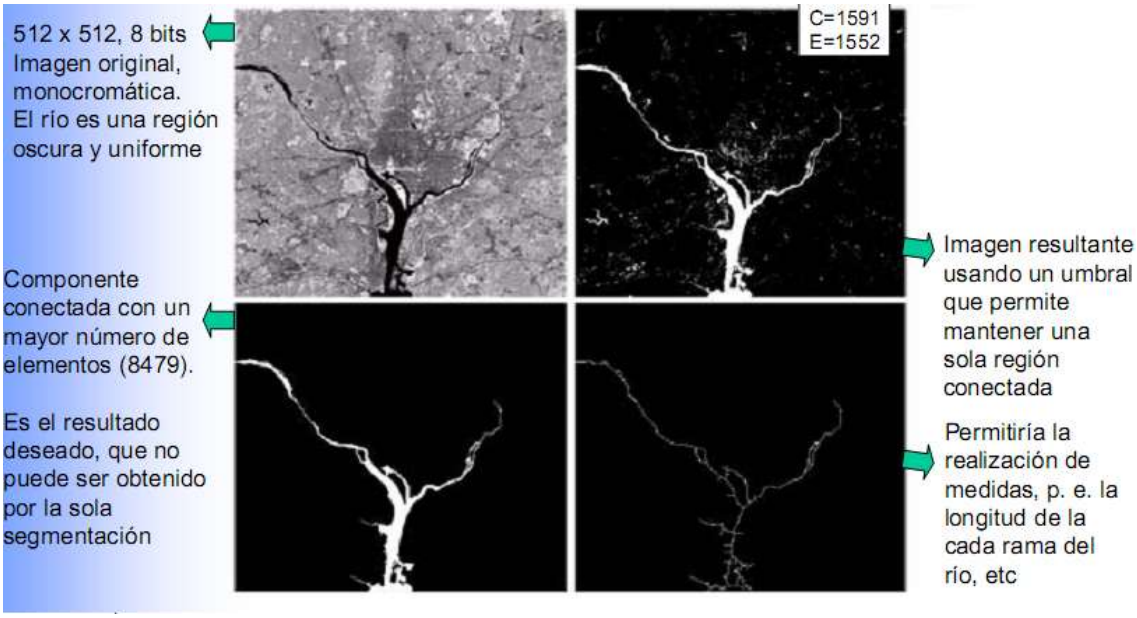


Figura 49.- a) Imagen infrarroja de Washington D.C. b) Imagen umbralizada. c) La componente conectada más grande. d) Esqueletización.

- Texturas: los descriptores de textura proporcionan información de propiedades tales como: suavidad, tosquedad (coarseness) y regularidad. Existen descriptores de texturas estadísticos (suavizado, burdo, granulado), estructurales (basados en el concepto de que una textura primitiva puede ser usada para formar estructuras más complejas mediante el uso de reglas concretas) y espectrales (se basan en propiedades del espectro de Fourier (p.e. medida de periodicidades globales).

DESCRIPCIÓN MEDIANTE COMPONENTES PRINCIPALES (CP)

El objetivo de la descomposición en CP es resumir un grupo de variables en un conjunto más pequeño, sin perder por ello una parte significativa de la información original.

Se utiliza la matriz de covarianza de los datos, que indica la relación entre distintas características.

Los autovectores de la matriz de covarianza marcan la orientación preferente de la distribución, indican las direcciones de mayor dispersión.

Los autovalores de la matriz de covarianza indican la varianza en la dirección de los correspondientes autovectores, indican la proporción de la información original que contiene esa nueva característica.

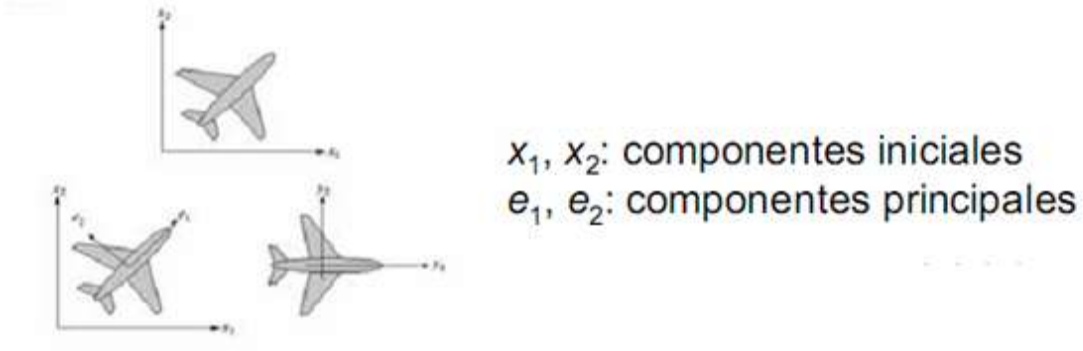


Figura 50.- Componentes principales de un objeto.

Las CP del color es uno de los métodos más usados.

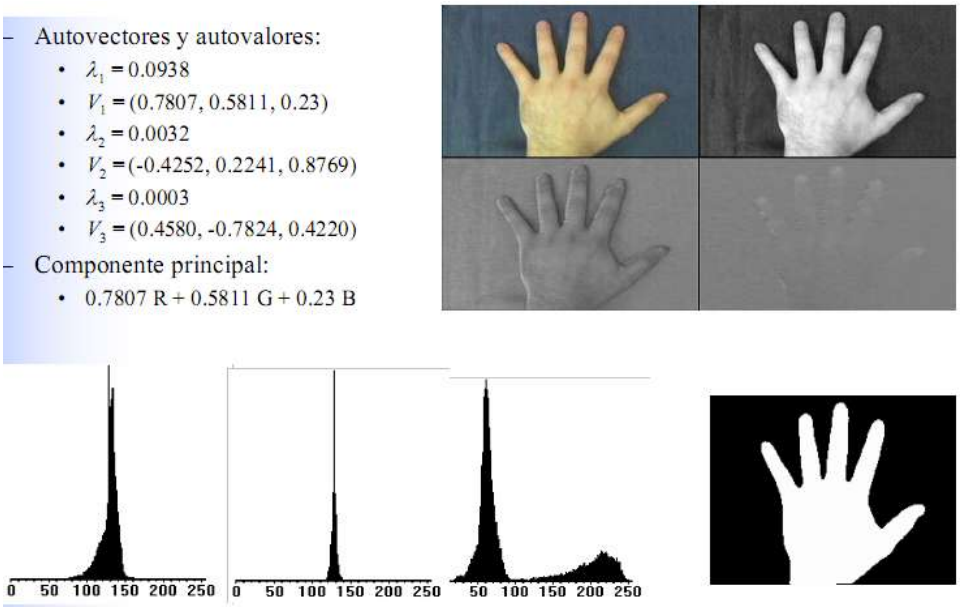


Figura 51.- Ejemplo de uso de las componentes principales del color.

CONCLUSIONES AL CAPÍTULO

A lo largo del capítulo se ha introducido multitud de conceptos útiles a la hora de generar características descriptoras de los objetos. Sin embargo, cada uno de estos enfoques por separado suele resultar insuficiente para describir los objetos de la mayoría de los problemas reales. Por ello, suele usarse combinaciones de varios de los métodos propuestos y también suele realizarse modificaciones para ajustar estos métodos al problema particular que se trate.

BIBLIOGRAFÍA DEL CAPÍTULO

[GW93], [AK89], [Ru95] y [Pra91].

2.1.6 SEGUIMIENTO DE OBJETOS EN MOVIMIENTO

Cuando tratamos con una fuente de video, que es diferente a imágenes individuales, en ocasiones tenemos un objeto u objetos en particular que queremos seguir a través del campo de visión. En capítulos anteriores vimos como aislar una forma en particular, ahora lo que queremos hacer es comprender el movimiento de dicho objeto, una cuestión que tiene dos componentes principales: *identificación y modelado*.

La identificación consiste en buscar el objeto en el que estamos interesados en un frame dentro de una secuencia. Técnicas como los momentos o los histogramas de color vistos en otros capítulos nos ayudaran a identificar el objeto que estamos buscando. Seguir cosas que todavía no hemos identificado es un problema. El seguimiento de objetos no identificados es importante cuando queremos determinar algo con su movimiento o cuando el movimiento del propio objeto es lo que le hace interesante. Las técnicas de seguimiento de objetos no identificados normalmente involucra el seguimiento de puntos clave muy significantes. En este capítulo vamos a estudiar dos métodos de seguimiento de objetos: el algoritmo de *Lucas-Kanade* y el de *Horn-Schunk*, que representan los flujos ópticos no compactos y compactos respectivamente.

La segunda componente, el modelado, nos ayuda con técnicas para conseguir una medida ruidosa de la posición actual del objeto. Algunas de las técnicas matemáticas más potentes han sido desarrolladas estimando la trayectoria de un objeto medido de una manera ruidosa. Estos métodos son aplicables a modelos bidimensionales y tridimensionales de los objetos y sus posiciones.

BÚSQUEDA DE "ESQUINAS" (CORNERS)

Hay varios tipos de características locales que se pueden seguir. Obviamente si cogemos un punto en una pared blanca enorme no va a ser fácil encontrar ese mismo punto en el siguiente frame de un video. Si todos los puntos de la pared son idénticos o muy similares no vamos a tener mucha suerte siguiendo ese punto en una secuencia de frames. Por otro lado, si cogemos un punto que es único tenemos muchas probabilidades de encontrar ese punto otra vez. En la práctica, el punto o característica seleccionada debe ser único, o casi único, y debe ser parametrizable de manera que pueda ser comparado con otros puntos en otra imagen.

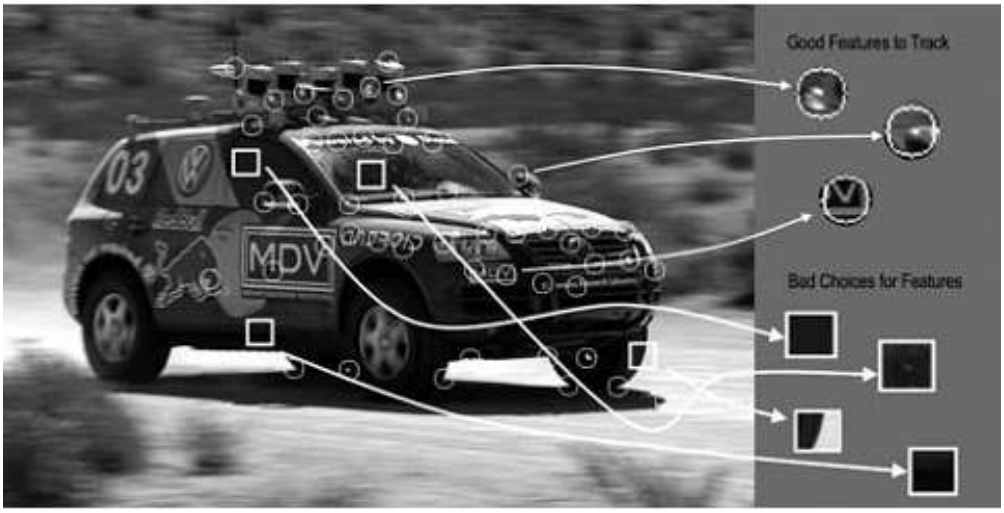


Figura 52.- Los puntos incluidos en círculos son óptimos para su seguimiento, los que están en cuadrados son malas opciones.

Si cogemos cambios significantes y los observamos en dos direcciones ortogonales podemos esperar que ese punto sea más susceptible de ser único. Por esta razón, a estas características que se pueden seguir se las llaman esquinas. Las esquinas, que nos son bordes, son puntos que contienen suficiente información como para ser encontrados frame a frame.

FLUJO ÓPTICO

Imaginemos que queremos apreciar el movimiento entre dos frames (o una secuencia de frames) sin tener otro conocimiento del contenido de esa imagen. Normalmente, el propio movimiento es el que nos indica que algo interesante está pasando. El flujo óptico puede verse en la figura 53.

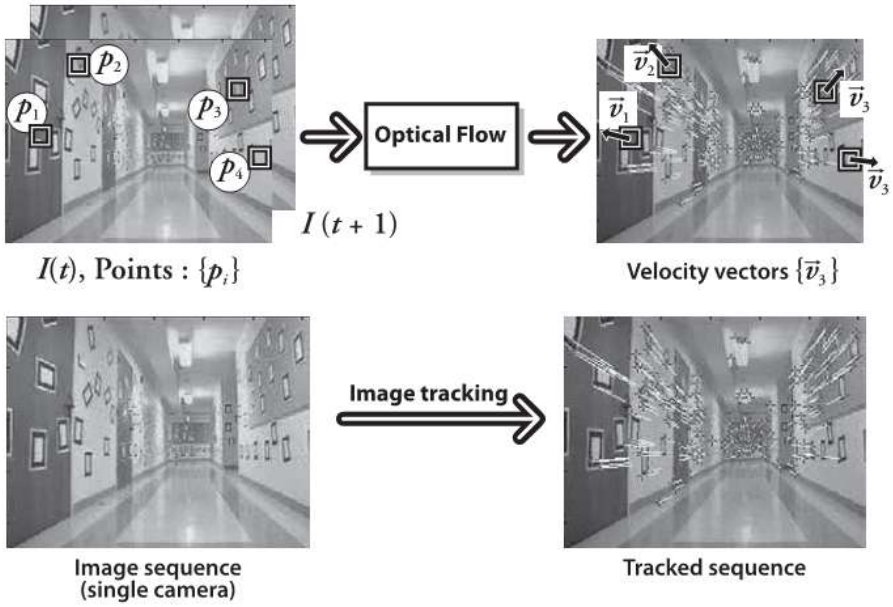


Figura 53.- Flujo óptico: a) los objetivos, b) son seguidos en todo momento y su movimiento es transformado en vectores de velocidad. c) Imagen de un hall. d) vectores de flujo originados debido al movimiento de la cámara por el hall.

Podemos asociar la velocidad con cada pixel, algunos desplazamientos se representan mediante la distancia a un pixel que se ha movido entre dos frames seguidos. En el flujo óptico compacto se asocia una velocidad con cada pixel de la imagen. El método de Horn-Schunk se basa en el cálculo de un campo de velocidad. Se pueden buscar ventanas alrededor del pixel en un frame y tratar de encontrarlas en el siguiente, a esto se le llama búsqueda de bloque. Ambas rutinas son consideradas como *técnicas de seguimiento compactas*.

En la práctica no es fácil calcular flujos ópticos compactos. Considerando el movimiento en una hoja blanca de papel, algunos de los pixeles blancos del frame anterior serán blancos de nuevo en el frame actual, solo los bordes pueden cambiar. El resultado es que estos métodos compactos deben tener alguna técnica de interpolación entre puntos para que sean seguidos más fácilmente. Estas dificultades muestran la alta complejidad y el alto coste computacional de los flujos ópticos compactos.

La otra alternativa son los flujos ópticos no compactos. Los algoritmos de esta naturaleza tratan de buscar puntos óptimos para el seguimiento, como las esquinas, una vez encontrados estos, el seguimiento será relativamente fiable y robusto. El coste computacional de estas técnicas es mucho menor que las anteriores por lo que el método anteriormente estudiado queda relegado al mero interés académico. La técnica más famosa es la de Lucas-Kanade (LK).

Una característica importante acerca de la cámara que elegiremos para capturar las imágenes es su velocidad de captura. Si un objeto se mueve rápidamente y la cámara es capaz de captar pocos frames por segundo (fps), se perderá el objeto. La razón es muy simple, la cámara toma una imagen del objeto y en la siguiente captura el objeto ya está muy alejado del lugar anterior, con lo cual no es capaz de encontrarlo. En cambio con una cámara con mayor número de fps en ese mismo espacio de tiempo habrá tomado más imágenes que en el caso anterior y el movimiento del objeto será menos brusco para el sistema.

EL MÉTODO DE LUCAS-KANADE

El algoritmo de Lucas-Kanade (LK) fue propuesto en 1981 como un intento de producir resultados compactos. El algoritmo puede ser aplicado en un contexto no compacto porque utiliza información local derivada de pequeñas ventanas que envuelven los puntos de interés, en contraposición a la naturaleza global del algoritmo de Horn y Schunk. La desventaja de usar pequeñas ventanas locales en LK es que ante grandes movimientos se pueden mover los puntos fuera de esta ventana y por tanto sería imposible para el algoritmo volver a encontrarlos. Este problema se soluciona con el *algoritmo de LK piramidal*, cuyo seguimiento comienza en el nivel más alto de una imagen piramidal (pocos detalles) y trabaja disminuyendo los niveles (aumento de detalles). El seguimiento a través de imágenes piramidales permite que los grandes movimientos sean capturados por las ventanas locales.

¿CÓMO TRABAJA EL ALGORITMO DE LK?

La idea básica de este algoritmo se basa en tres asunciones:

- Brillo constante. Un pixel de la imagen o un objeto de una escena no puede cambiar de apariencia (en la medida de lo posible) en su movimiento. Para una imagen en escala de grises esto significa que el brillo de un pixel no debe cambiar.
- Persistencia temporal de "pequeños movimientos". El movimiento de una zona de la imagen cambia lentamente en el tiempo. En la práctica esto significa que el incremento temporal es demasiado rápido para la escala del movimiento, el objeto no se mueve mucho entre frames.
- Coherencia espacial. Los puntos vecinos de una escena pertenecen a la misma superficie, que tiene un movimiento similar, y proyecta los puntos cercanos en el plano de la imagen.

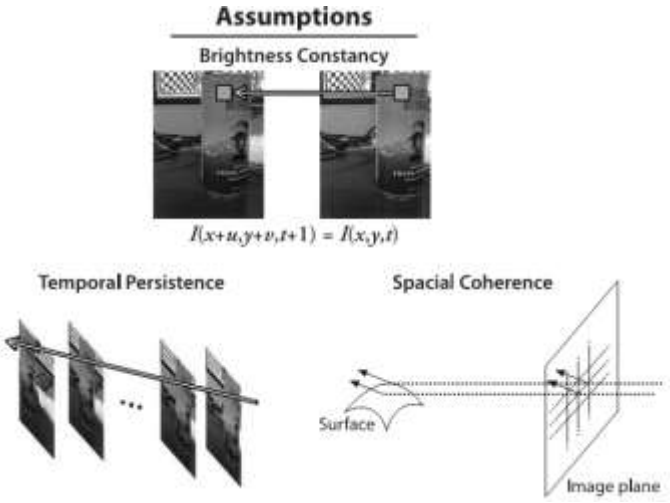


Figura 54.- Asunciones del algoritmo de Lucas-Kanade.

MÉTODO DE HORN-SCHUNK

El método de Horn-Schunk fue creado en 1981. Esta técnica es una de las primeras que usaban la asunción de la constancia en el brillo y derivaban las ecuaciones básicas del brillo constante. La solución de estas ecuaciones se hicieron bajo la hipótesis de suavizar la fuerza de las velocidades en el eje "x" e "y". Esta fuerza es derivada minimizando la laplaciana de las componentes de velocidad del flujo óptico. Esto sirve para penalizar aquellas regiones en las cuales el flujo está cambiando de magnitud. Como en el caso del algoritmo de LK se utilizan las iteraciones para resolver las ecuaciones diferenciales.

BIBLIOGRAFÍA DEL CAPÍTULO

[BK08].

2.1.7 RECONOCIMIENTO DE OBJETOS

En este capítulo se estudiarán diferentes algoritmos que permiten clasificar los elementos que aparecen dentro de una escena para poder entenderla. Los algoritmos de clasificación tienen la misión de distinguir entre objetos diferentes de un conjunto predefinido llamado *universo de trabajo*. Normalmente, el universo de trabajo se considera dividido en una colección K de *clases (α1,α2...αK),* perteneciendo los diferentes objetos a algunas de estas clases.

En este capítulo se estudiarán diferentes métodos que permiten determinar, de manera automática, en qué clase se encuentra un objeto de un universo de trabajo. Estos métodos se conocen como clasificadores.

CARACTERÍSTICAS DISCRIMINANTES

Para poder realizar el reconocimiento automático de los objetos se realiza una transformación que convierte un objeto del universo de trabajo en un vector X cuyas N componentes se llaman *características discriminantes* o *rasgos*.

Estas características deben permitir discriminar a qué clases puede pertenecer cualquier objeto del universo de trabajo.

$$X = (x_1, x_2, \ldots x_N) \quad \text{con} \quad N \in \mathcal{N} \quad \text{y} \quad x_i \in \Re \quad \forall i = 1 \ldots N$$

El valor del vector de características para un objeto concreto se conoce como patrón. Es decir, un patrón es una instancia particular de un vector de características determinado.

La determinación de las N características discriminantes es un proceso difícil que suele requerir del uso de la imaginación. En general, suelen usarse características como los momentos de los objetos a reconocer, alguna transformación de los mismos (Fourier, cosenos...), las propias imágenes, o cualquier característica que se pueda obtener de los objetos mediante algún procedimiento algorítmico.

Una vez determinadas las características discriminantes para un problema concreto, la clasificación de un objeto comienza por la obtención de su patrón. El siguiente paso consiste en determinar la proximidad o grado de pertenencia de este patrón a cada una de las clases existentes. A este efecto se definen las *funciones discriminantes* o *funciones de decisión* como aquellas funciones que asignan a un patrón un grado de semejanza respecto a cada una de las diferentes clases.

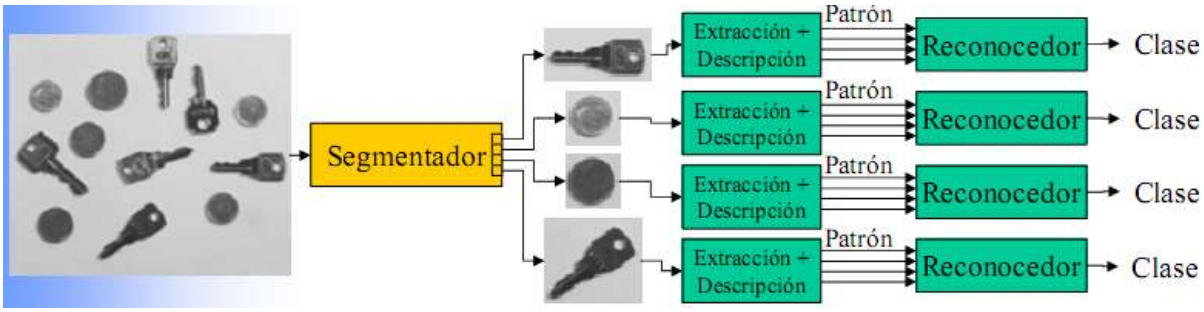


Figura 55.- Esquema de un sistema de visión por computador.

LA MUESTRA DE APRENDIZAJE

Para poder realizar el cálculo de las funciones discriminantes suele precisarse la existencia de un conjunto de patrones similares a los que se desea reconocer, que se denomina *conjunto de aprendizaje* o *conjunto de entrenamiento*. Los patrones de este conjunto se utilizan a modo de modelos para crear la función discriminante que clasificará correctamente los patrones del universo de trabajo. Por eso, el conjunto de aprendizaje debe estar constituido por un subconjunto representativo del universo de trabajo.

Cuando la muestra es abundante suele crearse otro conjunto con ella. Este segundo conjunto se utiliza para probar los resultados de las funciones discriminantes calculadas, y se conoce como conjunto de test. Es importante que el conjunto de aprendizaje y el de test sean independientes. Como norma general, en el caso de un universo de trabajo grande, la independencia queda asegurada si el conjunto de aprendizaje y el de test no tienen elementos en común. Esta independencia permite cierta confianza empírica en que el clasificador desarrollado posee la propiedad de generalización. Esta propiedad garantiza que un sistema clasifica correctamente patrones que no ha visto durante el proceso de cálculo de funciones discriminantes.

Si una vez construidos los clasificadores se prueban, usando el conjunto de test, y se obtienen unos resultados deficientes, debe descartarse el conjunto de test y volver a comenzar de nuevo con un nuevo clasificador y nuevos conjuntos.

Normalmente se recomienda tomar alrededor del 65% de la muestra para construir el conjunto de aprendizaje y un 35% para el conjunto de test.

CRITERIOS PARA LA SELECCIÓN DE CARACTERÍSTICAS

En general se busca el conjunto mínimo de características que permiten determinar de manera unívoca a qué clase pertenecen todos los objetos del universo de trabajo. Una elección mala de las características discriminantes puede hacer que el sistema sea innecesariamente caro y lento, o que sea imposible construir un clasificador para resolver un problema utilizando tales características.

Se pueden exigir cinco propiedades que deben poseer las características que se seleccionen: *economía, velocidad, fiabilidad, capacidad discriminante e independencia* con respecto a otras características.

Economía

El mecanismo preciso para el cálculo o la obtención de las características discriminantes (sensores, etc..) debe tener un coste razonable.

Velocidad

El tiempo de cálculo no debe superar el umbral que lo haga inviable.

Independencia

Las características no deben estar correladas entre ellas. Una característica que depende fuertemente del resto no añade información discriminante y por tanto puede eliminarse sin que esto suponga ninguna pérdida de capacidad discriminante.

Fiabilidad

La fiabilidad implica que objetos de la misma clase deben tener vectores de características con valores numéricos similares. Esto se cumple si los vectores de características de una clase tienen poca dispersión. La dispersión se puede medir sobre la diagonal de la matriz de covarianzas. Cuanto mayores son los valores de la diagonal, mayor es la dispersión.

Capacidad discriminante

La capacidad discriminante de una característica determinada se puede describir como una propiedad que asegura que patrones de clases distintas tienen valores numéricos claramente diferenciados.

PRINCIPALES ALGORITMOS DE CLASIFICACIÓN

A continuación se detallan algunos de los algoritmos de clasificación más utilizados en el campo de la visión por computador:

KNN (K- VECINOS MÁS CERCANOS)

El método ***k*-nn** es un método de clasificación supervisada (Aprendizaje, estimación basada en un conjunto de entrenamiento y prototipos) que sirve para estimar la función de densidad $F(x / Cj)$ de las predictoras x por cada clase C_j.

Este es un método de clasificación no paramétrico, que estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento x pertenezca a la clase C_j a partir de la información proporcionada por el conjunto de prototipos. En el proceso de aprendizaje no se hace ninguna suposición acerca de la distribución de las variables predictoras.

En el reconocimiento de patrones, el algoritmo *k*-nn es usado como método de clasificación de objetos (elementos) basado en un entrenamiento mediante ejemplos cercanos en el espacio de los elementos. *k*-nn es un tipo de "Lazy Learning" (en), donde la función se aproxima solo localmente y todo el computo es diferido a la clasificación.

DTREE (ÁRBOLES DE DECISIÓN)

Un **árbol de decisión** es un modelo de predicción utilizado en el ámbito de la inteligencia artificial. Dada una base de datos se construyen diagramas de construcciones lógicas, muy similares a los sistemas de predicción basados en reglas. Estos sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema.

Un árbol de decisión lleva a cabo un test a medida que este se recorre hacia las hojas para alcanzar así una decisión. El árbol de decisión suele contener nodos internos, nodos de probabilidad, nodos hojas y arcos. Un nodo interno contiene un test sobre algún valor de una de las propiedades. Un nodo de probabilidad indica que debe ocurrir un evento aleatorio de acuerdo a la naturaleza del problema, este tipo de nodos es redondo, los demás son cuadrados. Un nodo hoja representa el valor que devolverá el árbol de decisión y finalmente las ramas brindan los posibles caminos que se tienen de acuerdo a la decisión tomada.

RTREE (ÁRBOLES ALEATORIOS)

El algoritmo de los árboles aleatorios trata de resolver problemas de clasificación y de regresión. Los árboles aleatorios son una colección de predictores árbol llamados “bosque”. La clasificación se realiza de la siguiente manera: el algoritmo coge el vector de entradas y lo clasifica mediante todos los árboles del “bosque” y como salida nos devuelve la clase a la que pertenece (la que ha obtenido “la mayoría de votos”). En el caso de la regresión, la respuesta del clasificador es la media de las respuestas de todos los árboles del “bosque”.

SVM (MÁQUINA DE VECTORES DE SOPORTE)

Las **máquinas de soporte vectorial** o **máquinas de vectores de soporte** (*Support Vector Machines*, SVMs) son un conjunto de algoritmos desarrollados recientemente por Vladimir Vapnik y su equipo en los laboratorios AT&T. Pertenecen a la familia de los clasificadores lineales puesto que inducen separadores lineales o hiperplanos en espacios de características de muy alta dimensionalidad (introducidos por funciones núcleo o kernel) con un sesgo inductivo muy particular (maximización del margen)

Inicialmente se usaron para problemas de clasificación binaria, pero después se ha extendido su uso a problemas de regresión, agrupamiento, clasificación multiclase, regresión ordinal, y se está trabajando en la búsqueda de resolver problemas más complejos (árboles y grafos).

NN (REDES NEURONALES)

Las **redes de neuronas artificiales** (denominadas habitualmente como **RNA** o en inglés como: "ANN") son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los

animales. Se trata de un sistema de interconexión de neuronas en una red que colabora para producir un estímulo de salida. En inteligencia artificial es frecuente referirse a ellas como **redes de neuronas** o **redes neuronales**.

NBC (CLASIFICADOR BAYESIANO)

Un clasificador **Bayesiano** es un clasificador probabilístico que se basa en aplicar el Teorema de Bayes. La función de distribución de los datos de asume que es gaussiana, una componente por clase. Durante el entrenamiento se calculan los vectores de medias y las matrices de covarianza de las clases y se usan para realizar la predicción.

EM (ESPERANZA-MAXIMIZACIÓN)

El algoritmo esperanza-maximización o algoritmo **EM** se usa en estadística para encontrar estimadores de máxima verosimilitud de parámetros en modelos probabilísticos que dependen de variables no observables. El algoritmo EM alterna pasos de esperanza (paso E), donde se computa la esperanza de la verosimilitud mediante la inclusión de variables latentes como si fueran observables, y un paso de maximización (paso M), donde se computan estimadores de máxima verosimilitud de los parámetros mediante la maximización de la verosimilitud esperada del paso E. Los parámetros que se encuentran en el paso M se usan para comenzar el paso E siguiente, y así el proceso se repite.

(K-MEANS) ALGORITMO DE LAS K-MEDIAS

El *algoritmo k-medias* permite determinar la posición de k centroides que distribuyan de manera equitativa un conjunto de patrones. Debe notarse que, a diferencia del algoritmo anterior, este algoritmo tiene la particularidad de necesitar conocer a priori el número k de clases existentes.

La Figura 58 muestra un ejemplo de aplicación del algoritmo de k-medias sobre un conjunto de patrones formados por dos características (por lo que se pueden presentar en un plano). El problema es tal que inicialmente no se conoce qué patrones pertenecen a cada una de las clases, pero se sabe que la muestra está dividida en dos clases, por lo que se aplica el algoritmo de k-medias con k=2. En el instante t=1 se toman como centroides dos patrones al azar. En la figura se muestra en gris la zona en la que están los patrones que están más cerca de ese centroide.

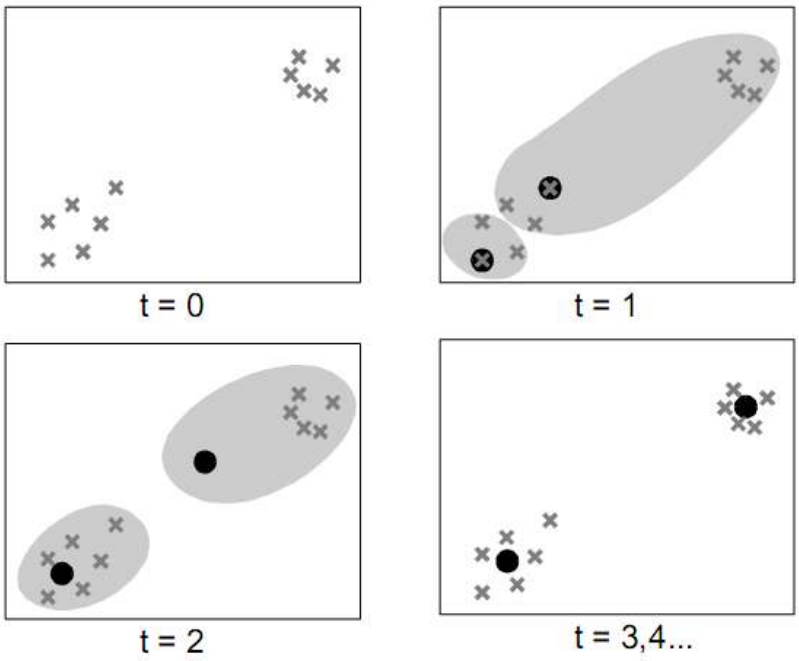


Figura 58.- Ilustración de la posición de los centroides (puntos negros) al aplicar el algoritmo de k-medias (con k=2), durante varias iteraciones para una muestra compuesta de vectores bidimensionales (cruces grises).

Serán los patrones de cada una de las zonas los que se usen para calcular el nuevo centroide en la siguiente iteración. En los instantes sucesivos se aprecia como los centroides van viajando hasta su ubicación definitiva. En la tercera iteración los centroides alcanzan una posición estable que no cambiará en sucesivas iteraciones, por lo que el algoritmo finaliza.

Este algoritmo goza de una amplia difusión debido a que es simple, a que suele ofrecer unos resultados fiables, y a que no depende de ningún umbral heurístico. Tiene la desventaja de que es preciso conocer de antemano el número de clases en que se divide la muestra. Aunque esto, en las ocasiones en que se dispone de tal dato, es más una ventaja que un inconveniente. También debe observarse que la solución obtenida puede ser dependiente de la elección que se haga de los patrones iniciales, por lo que debe ponerse cuidado en este punto. Esta variación del resultado dependiendo de los patrones iniciales ocurre sobre todo cuando la distinción entre clases no está muy clara, o cuando no se estima adecuadamente el número de clases.

(HMM) MODELO OCULTO DE MÁRKOV

Un **modelo oculto de Márkov** o **HMM** (por sus siglas del inglés, *Hidden Markov Model*) es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Márkov de parámetros desconocidos. El objetivo es determinar los parámetros desconocidos (u *ocultos*, de ahí el nombre) de dicha cadena a partir de los parámetros observables. Los parámetros extraídos se

pueden emplear para llevar a cabo sucesivos análisis, por ejemplo en aplicaciones de reconocimiento de patrones. Un HMM se puede considerar como la red bayesiana dinámica más simple.

Los modelos ocultos de Márkov son especialmente aplicados a reconocimiento de formas temporales, como reconocimiento del habla, de escritura manual, de gestos, etiquetado gramatical o en bioinformática.

CONCLUSIONES AL CAPÍTULO

La experiencia de los autores en los sistemas de visión artificial desarrollados demuestra que el usuario final suele ser poco comprensivo con los errores de la máquina en la clasificación, sobre todo en aquellos casos en los que el error es poco justificable desde el punto de vista humano. Por ello casi siempre es mejor aplicar diferentes enfoques de preprocesamiento, segmentación y clasificación y obtener tantos resultados independientes como se pueda. Estos resultados pueden combinarse a su vez mediante otros clasificadores. En el peor de los casos se puede decidir que el sistema, ante un patrón dudoso, no ofrezca resultado, a fin de que ofrezca fiabilidad a los usuarios finales.

Los algoritmos explicados en esta sección están implementados en la librería de procesamiento de imágenes OpenCV y pueden ser utilizados en la aplicación.

BIBLIOGRAFÍA DEL CAPÍTULO

[GW93], [JKS95], [Loo97], [Mar93] y [Wi].

2.2 LIBRERÍAS PARA EL PROCESAMIENTO DE IMÁGENES

Existen multitud de librerías creadas en varios lenguajes de programación que nos proporcionan las herramientas necesarias para crear aplicaciones de visión por computador. En este capítulo vamos a ver algunas de las alternativas más usadas: *JavaVis, OpenCV, MFMS, IVT* y el *Image Processing Toolbox de MATLAB.*

JAVAVIS

JavaVis es una librería de desarrollo de algoritmos y métodos para visión computacional (visión artificial) en Java. En esta librería se integran clases y

métodos para dar soporte a los métodos y algoritmos que deseemos implementar. Por un lado se han desarrollado las clases de soporte básicas para: definir el tipo de datos imagen, manejar lectura y escritura de ficheros, usar una interfaz gráfica que nos permita ver el resultado de aplicar determinados algoritmos. En base a estas clases básicas, se han desarrollado distintos métodos "clásicos" y útiles de visión computacional. De esta forma, disponemos de un conjunto de métodos de los que podemos hacer uso para desarrollar métodos más complejos.

JavaVis se empezó a desarrollar siguiendo la misma filosofía de trabajo de Vista. Vista fue una librería de visión escrita en C estándar que pretendía simular la programación orientada a objetos. Las características de Vista eran muy atractivas para el desarrollo de algoritmos en visión, pero su creador abandonó el proyecto. El objetivo era crear una librería similar a Vista con un lenguaje plenamente orientado a objetos. La mayoría de las características de JavaVis se fundamentan en el funcionamiento de Vista.

Se trata de una librería de uso libre y está disponible para los sistemas operativos Windows y Linux.

OPENCV

OpenCV (*Open Computer Vision*) es una librería de software libre de visión por computador que incluye cuatro componentes diferentes para la visión por computador en tiempo real. Ha sido creada por un equipo de desarrolladores de todo el mundo. El primer componente es CxCore, el cual nos proporciona los tipos genéricos que serán usados por las demás librerías como pueden ser arboles, listas, colas, secuencias, imágenes y las funciones que operan sobre estos. El segundo componente es CvReference que incluye todas las funciones de análisis y procesamiento. El tercer componente es la librería CvAux que contiene componentes experimentales y obsoletos del proyecto OpenCV. El último componente es la librería HighGUI que nos proporciona los elementos necesarios para la captura, presentación y almacenamiento de imágenes.

El componente HighGUI viene como una librería adicional. Esta librería nos proporciona una extensa lista de interfaces de captura. No provee herramientas para crear diferentes hilos ni flujos de datos, la adquisición, procesamiento y presentación de la imagen debe ser explicita.

OpenCV es una fuente fantástica de implementaciones estandarizadas de las funciones de visión por computador. Es posible que un quinto componente específico para sistemas de visión por computador sea añadido a la colección de OpenCV.

Esta librería está disponible para los lenguajes de programación C++ y Python y para los sistemas operativos Windows, Linux y Mac OS.

MODULAR FLOW SCHEDULING MIDDLEWARE FRAMEWORK

El *Modular Flow Scheduling Middleware Framework (MFSM)* ha sido creado para aplicaciones interactivas. MFSM implementa una arquitectura de software SAI, este término es debido al hecho de que deriva del campo de la aplicación. Esta arquitectura es un modelo de procesamiento de flujos paralelos de datos genéricos. Los componentes de la arquitectura SAI son muy estrictos y están hechos de fuentes, celdas, flujos y pulsos. Las fuentes generan datos y las celdas crean pulsos de datos para ser enviados como flujos. Los flujos activos conectan las celdas o "centros de procesamiento" juntos, donde cada celda añade datos a las estructuras de pulsos y pasan a través de las celdas de flujos de bajada. Los flujos pasivos conectan fuentes a las celdas y retienen los datos persistentes en la aplicación. Los flujos activos representan los datos volátiles.

El software está organizado en cuatro módulos diferentes que están separados en paquetes. Estos forman el paquete MFSM y añaden funcionalidades específicas como la representación y procesamiento de imágenes. Tiene integradas tres librerías de algoritmos de procesamiento estandarizados a través de células que se usan para construir el sistema de visión. Otras librerías como OpenCV, Unicap y OpenGL se usan para proporcionar la representación, captura y presentación de las imágenes respectivamente.

MFSM es una librería de uso libre escrita para C++. Está disponible para Windows, Linux y Mac OS.

INTEGRATING VISION TOOLKIT

The *Integrating Vision Toolkit (IVT)* es un paquete de software que incluye clases para la captura, procesamiento, presentación, sincronización, creación de hilos y algoritmos de procesamiento. Tiene una interfaz genérica para cada tipo de componente que permite el uso de distintos interfaces sobre Windows y Linux con una interfaz común. Los formatos internos de imagen son RGB y escala de grises. Existen métodos para convertir entre el tipo de imagen de OpenCV y su propio tipo de imágenes. La mayoría del soporte del GUI usa las herramientas Qt.

IVT es otro ejemplo de aplicaciones de software de nacimiento reciente. Es un sistema que da pasos prometedores, pero carece de la capacidad para crear flujos de datos automáticos e hilos seguros. Estos requerimientos serán añadidos por su equipo de desarrollo. Se usan varias herramientas para la captura y representación.

IVT está disponible para Windows, Linux y Mac OS. Esta librería de software libre trabaja en C++.

IMAGE PROCESSING TOOLBOX DE MATLAB

Este Toolbox proporciona a MATLAB un conjunto de funciones que amplía las capacidades del producto para realizar desarrollo de aplicaciones y de nuevos algoritmos en el campo del proceso y análisis de imágenes. El entorno matemático y de creación de MATLAB es ideal para el procesado de imágenes, ya que estas imágenes son, al fin y al cabo, matrices.

El proceso de imágenes es un campo de trabajo absolutamente crucial para aquellos colectivos e industrias que estén trabajando en áreas como diagnóstico médico, astronomía, geofísica, ciencias medioambientales, análisis de datos en laboratorios, inspección industrial, etc. El Image Processing Toolbox entra dentro de la categoría de familias de funciones que, desde el entorno de trabajo de MATLAB, permitirá al profesional efectuar una exploración exhaustiva y desde un punto de vista matemático de las imágenes y gráficos que se deseen tratar o analizar. Además la herramienta vfm nos permitirá la captura de imágenes captadas por dispositivos instalados en la computadora tales como tarjetas captadoras y USB Webcams.

Algunas de las funciones más importantes incluidas dentro de este toolbox son las siguientes: análisis de imágenes y estadística, diseño de filtros y recuperación de imágenes, mejora de imágenes, operaciones morfológicas, definición de mapas de colores y modificación gráfica, operaciones geométricas, transformación de imágenes y proceso de bloques

MATLAB es un producto comercial que opera con los sistemas operativos Windows, Linux, Mac OS y Solaris.

BIBLIOGRAFÍA DEL CAPÍTULO

[Fra], [IVT], [MAT], [Open] y [JVis].

2.3 LENGUAJES DE PROGRAMACIÓN

En este apartado se presentaran algunos de los lenguajes de programación más usados y desarrollados: *C, C++, Java y Python.*

C

C es un lenguaje de programación creado en 1972 por Kenneth L. Thompson y Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix . C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

CARACTERÍSTICAS DE IMPORTANCIA

- Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de ficheros, proporcionadas por bibliotecas.
- Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado no llevado al extremo (permitiendo ciertas licencias rupturistas).
- Un sistema de tipos que impide operaciones sin sentido.
- Usa un lenguaje de preprocesado, el preprocesador de C, para tareas como definir macros e incluir múltiples ficheros de código fuente.
- Acceso a memoria de bajo nivel mediante el uso de punteros.
- Interrupciones al procesador con uniones.
- Un conjunto reducido de palabras clave.
- Por defecto, el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros.
- Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo.
- Tipos de datos agregados (`struct`) que permiten que datos relacionados (como un empleado, que tiene un id, un nombre y un salario) se combinen y se manipulen como un todo (en una única variable "empleado").

C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje *multiparadigma*.

Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.

C++ permite trabajar tanto a alto como a bajo nivel.

El nombre **C++** fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

JAVA

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

PYTHON

Python es un lenguaje de programación interpretado creado por Guido van Rossum en el año 1991.

Se compara habitualmente con Tcl, Perl, Scheme, Java y Ruby. En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. La última versión estable del lenguaje es la 3.1.

Python es considerado como la "oposición leal" a Perl, lenguaje con el cual mantiene una rivalidad amistosa. Los usuarios de Python consideran a éste mucho más limpio y elegante para programar.

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender Python). También hay módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, sockets y hasta interfaces a GUI (interfaz gráfica con el usuario) como Tk, GTK, Qt entre otros.

Python se utiliza como lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa.

CONCLUSIONES AL CAPÍTULO

Todos estos lenguajes son usados en visión por computador. Cada uno tiene ciertas ventajas sobre los demás que son aprovechadas por las diferentes librerías de visión artificial para diferentes fines. Por ejemplo, las librerías creadas sobre C++ tienen una mayor velocidad de procesamiento que las creadas para Java. Por otra parte las de Java tienen la posibilidad de funcionar en multitud de dispositivos y sistemas operativos diferentes gracias a la portabilidad que ofrece su *máquina virtual*.

BIBLIOGRAFÍA DEL CAPÍTULO

[Wi].

CAPÍTULO III

Materiales empleados

En esté tercer capítulo, se presenta al lector todos los materiales utilizados para la realización del proyecto, desde los dispositivos hardware y software utilizados hasta el entorno de desarrollo.

3.1 DISPOSITIVOS HARDWARE

Los dispositivos hardware usados para la realización de este PFC han sido:

- Ordenador Portátil: *Acer Aspire 5720G*
  - Intel Core 2 Duo processor T7300 (2.0 GHz, 800MHz FSB, 4MB L2 caché)
  - ATI Mobility Radeon HD 2300 (hasta 1024 MB compartidos)
  - 2GB DDR2 de memoria RAM
  - HDD 160GB
- Webcam Externa USB: *Hercules Blog Webcam.*

3.2 ELEMENTOS SOFTWARE

Para la realización del proyecto se han usado los siguientes programas:

- Sistema Operativo: Windows XP SP3.
- Entorno de desarrollo: eclipse.
- Cygwin.
- Librería de visión por computador OpenCV.
- Librería de manejo de lectura/escritura archivos XML: TinyXML.
- Procesador de textos: Microsoft Word 2007.
- Presentaciones: Microsoft PowerPoint 2007.
- Edición de imágenes: Paint y PhotoShop CS3.

3.3 JUSTIFICACIÓN DEL ENTORNO DE DESARROLLO

El entorno de desarrollo elegido debía cumplir varios requisitos y en función de estos requisitos se decidió elegir como lenguaje de programación C++ y como librería de visión por computador OpenCV.

Estas facilidades o requisitos mínimos que el entorno de desarrollo elegido debía proveer son:

a) Ser un entorno de desarrollo de última generación, probado, multiplataforma y con documentación suficiente sobre su funcionamiento.

b) Ofrecer un buen rendimiento en las operaciones de tratamiento digital de imágenes. Hoy en día, las aplicaciones de tratamiento digital de imágenes y todas las técnicas utilizadas para su procesamiento, constituyen duras pruebas al rendimiento de los sistemas informáticos que las albergan. Ello se debe, principalmente, a la complejidad de los algoritmos utilizados para su implementación, así como a la cantidad de información que ha de mantenerse en memoria, pensemos que una pequeña imagen de tamaño 300 x 200, la cual utiliza un byte para la representación de cada punto ocupa, aproximadamente, 60 Kbytes de almacenamiento. Y, en cuanto a la complejidad de los algoritmos, pensemos en el coste que supone un bucle que haga un recorrido por dicha imagen.

c) Poseer librerías de rutinas para las operaciones básicas de tratamiento digital de imágenes, lo cual nos permitiría centrar nuestro trabajo en los objetivos marcados, consiguiendo mejores resultados.

d) Ser abierto, es decir, que permita la comunicación con otros sistemas, ya sean gestores de bases de datos, facilidades aportadas por los Sistemas Operativos, rutinas implementadas en otros lenguajes, etc.

C++

Una de las principales razones de usar C++ es que es más rápido en tiempo de ejecución que otras alternativas ya que mientras que otros lenguajes como por ejemplo java necesitan una maquina virtual para ejecutar las aplicaciones, C++ no lo necesita. Además de esto nos ofrece otras características como son la programación orientada a objetos, programación estructurada, permite trabajar tanto a alto como a bajo nivel siendo muy óptimo, etc. Tiene una amplia documentación para el correcto y rápido uso de sus características.

OPENCV

No solo es importante elegir un buen lenguaje de programación sino que también existan buenas librerías de visión por computador que usen este lenguaje. En este caso al ser C++ uno de los lenguajes más usados existe una gran variedad de librerías de visión artificial.

OpenCV es una librería que proporciona todos aquellos requisitos que se necesitan para la realización de esta aplicación: funciones para la lectura y escritura de imágenes, videos, webcam y datos. Multitud de funciones para el procesado de imágenes, segmentación, descripción… Además nos proporciona técnicas de seguimiento y reconocimiento de objetos. El tema de la documentación es uno de los aspectos más positivos de esta librería ya que existe un libro, *Learning OpenCV*, en el que hay multitud de ejemplos y se explican las funciones y aspectos principales de la librería de una forma muy detallada.

Mientras que otras librerías como MFSM necesitan de otras librerías auxiliares para la captura y visión de las imágenes (incluso utilizan OpenCV), este software pose una librería gráfica llamada *HighGui* que nos proporciona estas funcionalidades.

OpenCV provee un marco de desarrollo fácil de utilizar y altamente eficiente debido a su capacidad para aprovechar las capacidades que proporcionan los procesadores multinúcleo. OpenCV puede además utilizar el sistema *Primitivas de Rendimiento Integradas de Intel*. Que es un conjunto de rutinas de bajo nivel específicas para procesadores Intel (IPP).

Estas características unidas al uso generalizado que tiene en el mundo de la visión por computador nos han llevado a elegir esta librería como un elemento necesario para el desarrollo del proyecto.

CAPÍTULO IV

Metodología empleada

En este capítulo se describen aquellas técnicas que se han usado para la realización de la aplicación.

4.1 SEGMENTACIÓN Y PROCESADO DE IMÁGENES

Para detectar los objetos que aparecen en la imagen se ha usado la *técnica de segmentación basada en movimiento.*

Como hemos visto anteriormente esta técnica es una herramienta potente de segmentación cuando tenemos un fondo estático y objetos en movimiento (como sucede en nuestro caso). Esta técnica se basa sencillamente en restar a la imagen del fondo con los objetos la imagen del fondo sin los objetos, como resultado tenemos una imagen solo con los objetos. Sin embargo no todo es tan sencillo ya que para que como resultado obtengamos exactamente los objetos deberíamos restar la imagen con objetos y la imagen sin objetos pero en el lugar donde antes estaban los objetos debería ser negro. De esta manera el fondo-fondo=0 y objetos-0(negro)=objetos (véase figura 1).

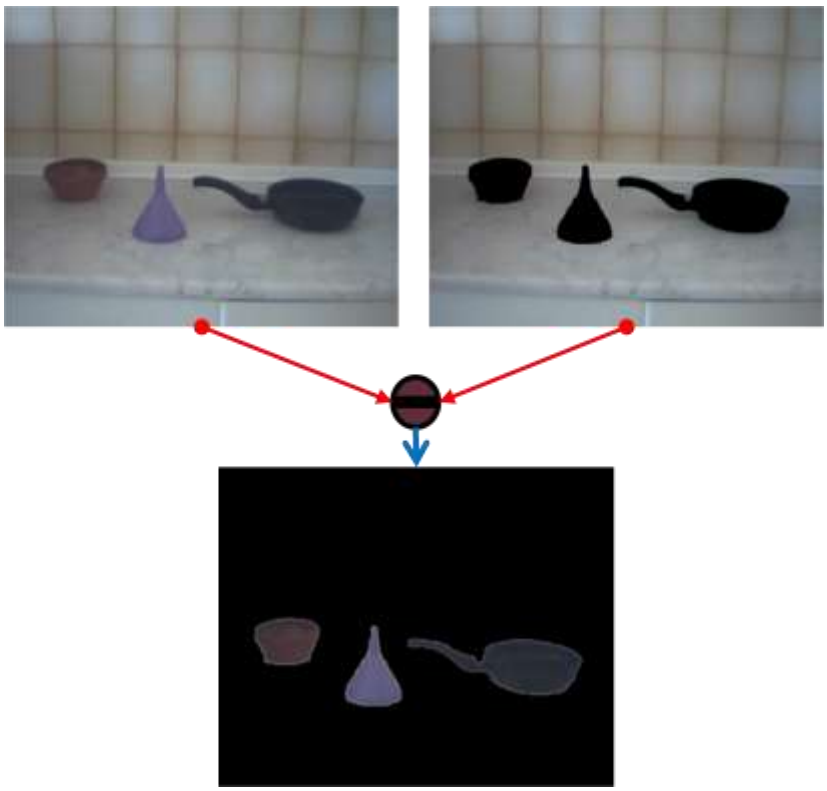


Figura 1.- Proceso para extraer los objetos del fondo.

En la realidad no sucede de esta manera porque las zonas de los objetos donde debería ser negro tenemos un fondo y además los objetos producen pequeñas sombras que también se verán reflejadas a la hora de restar las imágenes.

A partir de ahora para simplificar llamaremos *imagen del fondo* a la imagen sin objetos, *imagen actual* a la imagen del fondo con los objetos incluidos e *imagen diferencia* a la imagen resultado de la resta.

Los pasos que se han seguido para la segmentación de los objetos han sido los siguientes:

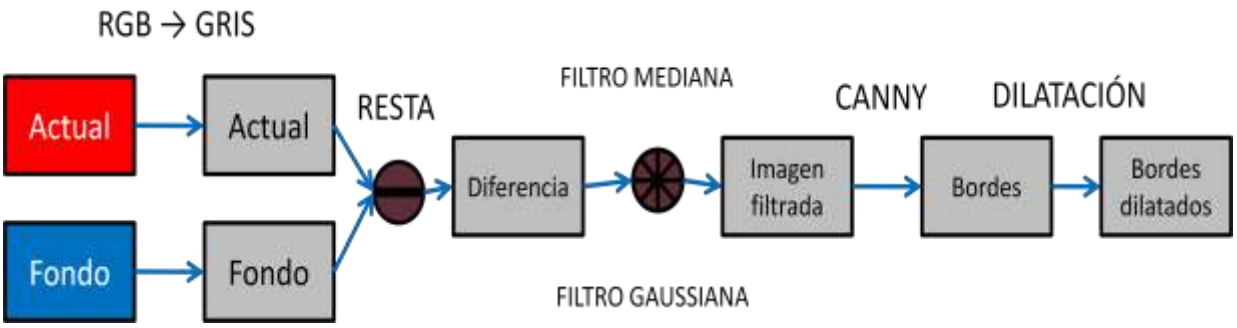


Figura 2.- Esquema general del proceso seguido para la segmentación.

OpenCV tiene implementadas las funciones necesarias para llevar a cabo los pasos explicados en la figura 2. A continuación se detallan estos procedimientos:

- En primer lugar convertimos las imágenes del fondo y la actual a blanco y negro. Para ello utilizamos la función de OpenCV **cvCvtColor**(**src,dst,CV_RGB2GRAY**) donde *src* es la imagen de entrada (en color), *dst* es la imagen de salida (en blanco y negro) y *CV_RGB2GRAY* indica la conversión entre formatos: de RGB a escala de grises.

- Ahora que ya tenemos las dos imágenes en escala de grises procedemos a restarlas mediante la función **cvAbsDiff**(**src1,src2,dst**) donde *src1* y *src2* son las dos imágenes que queremos restar y *dst* la imagen diferencia que nos da como resultado.

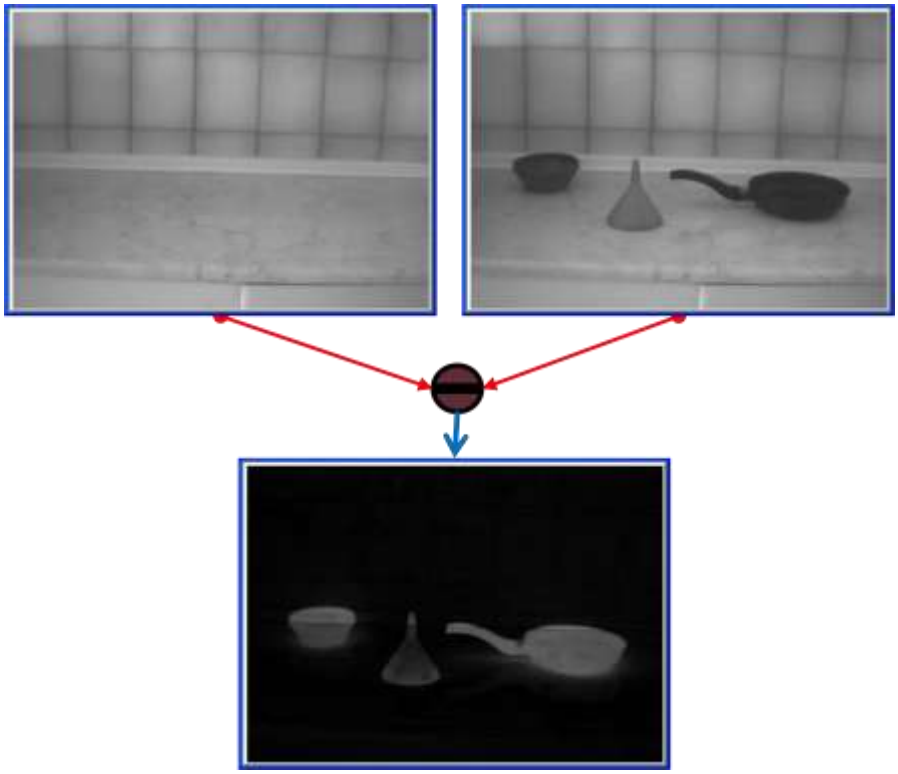


Figura 3.- Ejemplo del procedimiento seguido en la aplicación para la resta de imágenes.

Como podemos ver el resultado no son exactamente los objetos y aparecen pequeñas sombras que dificultan la tarea de segmentación pero de alguna manera hemos conseguido aislar los objetos del fondo.

- En el siguiente paso vamos a aplicar dos filtros a la imagen diferencia:

  Filtro de mediana de tamaño 3x3 píxeles:
  c**vSmooth**(**src,dst,CV_MEDIAN,3,3**).

  El tamaño del filtro debe ser suficientemente grande como para disminuir los detalles que aparecen en los objetos y eliminar impurezas en el fondo, pero no lo bastante grande como para eliminar por completo la forma del objeto. Experimentalmente se comprobó que 3x3 era el tamaño adecuado, ya que un tamaño menor no producía prácticamente cambios en la imagen y uno superior difuminaba la forma de los objetos.

  Filtro de gaussiana de tamaño 5x5 píxeles para difuminar los bordes que aparecen en la imagen. Con esto conseguimos que desaparezcan los bordes menos marcados como los de las sombras ya que al siguiente paso, que es la extracción de contornos, no queremos que estas aparezcan: **cvSmooth**(**src,dst,CV_GAUSSIAN_5x5**).

Realizando diferentes pruebas, se comprobó que el tamaño adecuado de este filtro era 5x5. Un tamaño menor, como 3x3, no era suficiente para difuminar las sombras y un tamaño superior, como 7x7, difuminaba incluso los objetos.

- A continuación procedemos a la extracción de los contornos de la imagen:

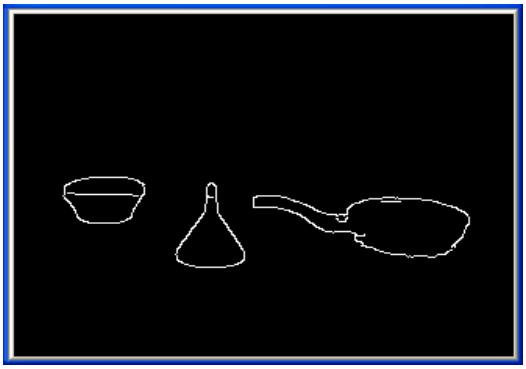


Figura 4.- Imagen con los contornos de los objetos.

La detección de bordes se realiza mediante la función **cvCanny** que, como su nombre indica, calcula los contornos mediante el método *canny*. A esta función se le pasa una imagen en escala de grises o en formato binario y devuelve otra imagen con los bordes calculados. La imagen resultado está en formato binario, toda la imagen es negra excepto los bordes que son blancos. También hay que introducir el tamaño de apertura, que indica el tamaño de *la matriz de Sobel*, en este caso 3x3. Este tamaño es el menor posible que nos permite OpenCV y con un tamaño superior se calculan demasiados bordes, apareciendo incluso bordes en el fondo.

- Estos bordes no son exactamente los bordes que nos interesan, queremos únicamente los bordes exteriores y con menor detalle, por ello vamos a realizar la dilatación de estos bordes.
  La dilatación es una de las operaciones morfológicas que vimos en el capítulo II. Para realizar la dilatación necesitamos crear un *elemento estructurante (EE):*
  IplConvKernel* element = **cvCreateStructuringElementEx** (**5 ,5, 0, 0, CV_SHAPE_ELLIPSE**);
  Mediante esta función creamos un elemento estructurante en forma de elipse de 5 filas x 5 columnas de píxeles y se lo asignamos a la variable **element**. La razón de la elección de este elemento es que al dilatar con él los bordes no tomarán un aspecto cuadrado, sino que quedarán con un aspecto similar al original como se puede ver en la figura 5. El tamaño de la elipse es de 5x5, lo suficiente para unir huecos entre los bordes que el

algoritmo no había sido capaz de detectar y no lo bastante grande como para unir el borde de un objeto con otro.

Una vez que tenemos el EE lo usaremos para la dilatación: **cvDilate(src,dst,element,3)**. Los parámetros de entrada de esta función son : imagen de entrada, imagen de salida, EE, numero de iteraciones.

Figura 5.- Imagen con los bordes dilatados.

Después de este último paso los objetos quedan bien diferenciados y queda la imagen preparada para pasar a la siguiente etapa: *representación y descripción.*

4.2 REPRESENTACIÓN Y DESCRIPCIÓN

Ya hemos detectado los objetos que aparecen en la imagen, ahora surge la necesidad de representarlos de alguna manera y posteriormente describirlos con varios parámetros para, a través de estos, poder realizar la clasificación.

REPRESENTACIÓN

Para representar los objetos se ha utilizado el contorno o frontera.

Para la librería OpenCV un contorno (CvSeq) no es más que una lista de puntos que definen una frontera. Para buscar contornos en una imagen es necesario que a la imagen se le haya aplicado el método *canny*, que hemos visto en capítulos anteriores, o en su defecto que sea una imagen binaria como es nuestro caso.

La función que realiza la búsqueda de contornos es:

cvFindContours(img ,almacen, &contorno, sizeof(CvContour), CV_RETR_EXTERNAL);

Donde *img* es la imagen de entrada. *almacen* es un objeto CvMemStorage, que es una entidad creada por OpenCV para manejar la asignación de memoria en objetos dinámicos. *&contorno* es un objeto secuencia (CvSeq) donde se

almacenará el contorno. *sizeof(CvContour)* indica el tamaño de la cabecera y *CV_RETR_EXTERNAL* indica a la función que solo almacene los contornos externos.

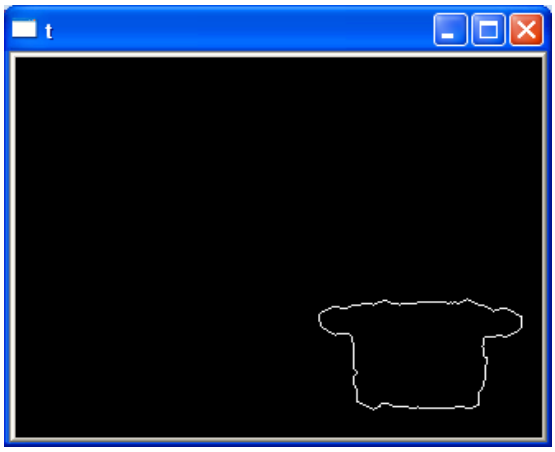


Figura 6.- Imagen a la que se le han extraído los bordes externos.

Para simplificar el contorno se le realiza una aproximación poligonal:

contorno = **cvApproxPoly** (**contorno, sizeof(CvContour), almacen, CV_POLY_APPROX_DP,2**);

Mediante esta función sustituimos la frontera hallada anteriormente y almacenada en el objeto *contorno* por otro contorno al que se le ha aplicado una aproximación poligonal. *CV_POLY_APPROX_DP* indica el método de aproximación, actualmente este método (Douglas-Peucker) es el único soportado por OpenCV. El *2* indica la precisión con que se realiza esta aproximación, es decir, cuanto mayor sea este número, más complejo será el polígono resultante (tendrá un mayor número de lados y vértices). Un valor inferior a 2 hace que el polígono sea demasiado simple, con muy pocos lados y perdiendo parte de su forma. Por otro lado, un valor superior a 2 hace que la aproximación tenga muchos lados y se parezca mucho al original, es como si en realidad no estuviéramos haciendo nada.

Al aplicar la función *FindContours* ya sabemos cuantos contornos ha encontrado en la imagen, este número de contornos indica también el número de objetos que hay en la imagen.

Esta función implementada por OpenCV nos permite dibujar el contorno hallado en una imagen:

cvDrawContours(**img,contorno,cvScalar(255),cvScalar(0),2,1**);

img es la imagen donde se va a dibujar el contorno y *contorno* la frontera a dibujar. *cvScalar(255)* indica que vamos a pintar el contorno de color blanco y

cvScalar(0) que vamos a pintar de negro el interior de los contornos. El número *2* indica que se pintan todos los contornos y el *1* el grosor de la línea.

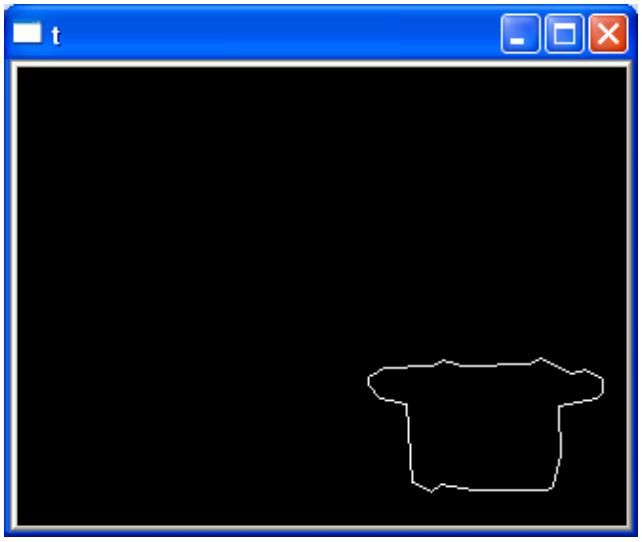


Figura 7.- Aproximación poligonal del contorno de la figura anterior.

DESCRIPCIÓN

Para describir los objetos vamos a utilizar como patrones algunos de los momentos más representativos de los contornos, la compacidad (perímetro^2/área) y el valor medio de los canales RGB del objeto. Para ello se ha creado una estructura que facilitará la obtención en todo momento de los patrones de cualquier objeto:

```
struct patrones{

	//compacidad=perimetro^2/área
	float compacidad;

	//Valor medio de la componente r del objeto
	double rMedio;

	//Valor medio de la componente g del objeto
	double gMedio;

	//Valor medio de la componente b del objeto
	double bMedio;

	//momento espacial m00
	double momEspacial;

	//momento central mu20
	double momCentral;

	//momento central mu02
	double momCentral2;


};
```

CvMoments es la entidad que usa OpenCV para manejar los momentos. *CvMoments momentos* es el objeto donde almacenaremos los momentos que posteriormente vamos a calcular.

Mediante la función **cvContourMoments**(**contorno,&momentos**); almacenamos en *&momentos* los momentos obtenidos de la frontera *contorno*.

Existen momentos espaciales y centrales, de los cuales nos interesan los siguientes:

- patron.momEspacial = momentos.m00;
- patron.momCentral = momentos.mu20;
- patron.momCentral2 = momentos.mu02;

La razón por la que se ha elegido el momento espacial 00 y los centrales 20 y 02 es porque son los que diferencian unos objetos de otros de una manera más clara. Para su elección se realizo el siguiente experimento: se representaron en una gráfica siete tipos de objetos diferentes (botellas de mistol, vasos, tazas, cazos, ollas, sartenes y botes). Cada objeto estaba representado por un punto (x,y), donde *x* es cualquiera de los tres momentos elegidos e *y* otro de los dos momentos restantes. En la gráfica se podía ver como los objetos aparecían divididos en varios grupos bien diferenciados siempre que se utilizasen algunos de estos tres momentos. De esta manera si se representasen en una grafica 3D donde (x,y,z) fuesen cada coordenada uno de los tres momentos elegidos aparecerían representados grupos de objetos diferenciados unívocamente. Por otro lado el resto de los momentos hacían una mala diferenciación entre los diferentes grupos de objetos, es decir, son características poco discriminantes. En las figuras mostradas a continuación se pueden ver los diferentes casos explicados:

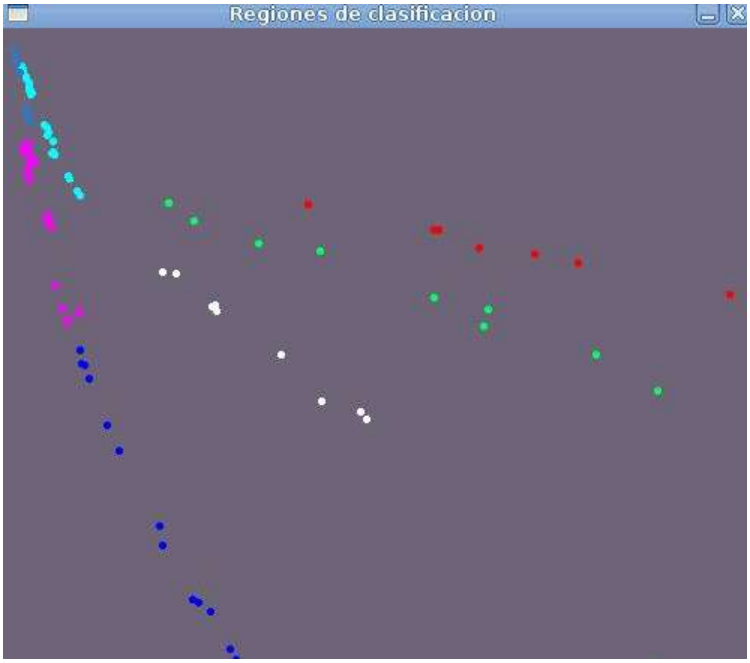


Figura 8.- Ejemplo de representación de objetos con dos momentos que diferencian bien los distintos grupos (momento central 20 y momento espacial 00). Cada color representa una serie de objetos de un mismo tipo.

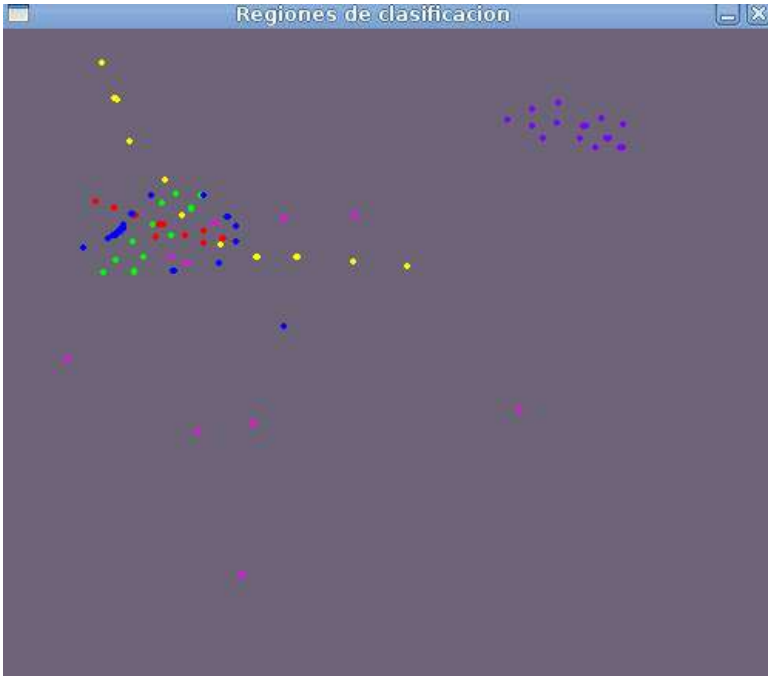


Figura 9.- Ejemplo de representación de objetos con dos momentos poco diferenciadores (momento espacial 10 y momento central 12). Los grupos de objetos son muy complicados de diferenciar.

La compacidad es el cociente entre el perímetro^2 y el área. La razón de usar este cociente en lugar del área o el perímetro es sencilla: si un objeto esta cerca de la

cámara será más grande en la imagen, por lo tanto su área y perímetro también lo serán. Si ocurre el caso contrario en el que están más lejos, su área y perímetro varían completamente. Sin embargo la compacidad varía menos frente a cambios de tamaño del objeto en la imagen.

- patrón.compacidad = compacidad;

Para poder representar los objetos por su color se han realizado los siguientes pasos:

- Copiamos la imagen del contorno en una imagen llamada *mask*:
  cvCopy(**imagen,mask**);
- Rellenamos el interior del contorno de blanco mediante la función:
  cvFloodFill(**mask,punto,cvScalar(255)**);
  Esta función rellena de un mismo color (*cvScalar(255) = blanco*) el interior de un contorno. *punto* es un pixel del interior de ese contorno y *mask* es la imagen donde se pinta.
  Gracias a esto hemos generado una máscara:

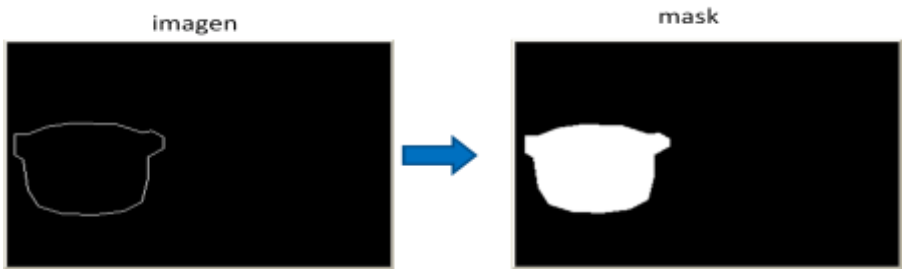


Figura 10.- Generación de una máscara a través de un contorno.

- Con esta mascara podemos quedarnos únicamente con el objeto y así medir el valor medio de sus componentes RGB:

Figura 11.- Obtención del objeto a través de la máscara.

CvScalar avg = **cvAvg**(**actual,mask**); almacena en *avg* los valores medios de los canales RGB de la imagen, *actual*, dentro de la máscara, *mask*.

- patron.rMedio = avg.val[2]; →R
- patron.gMedio = avg.val[1]; →G
- patron.bMedio = avg.val[0]; →B

Mediante un fichero de configuración podemos seleccionar, dentro de estos siete patrones, que patrones queremos que representen a los objetos para el entrenamiento y clasificación.

4.3 RECONOCIMIENTO DE OBJETOS

Una vez que ya tenemos una manera de describir los objetos podemos proceder a su clasificación. Para poder clasificarlos es necesario entrenar el clasificador, para ello se han obtenido varias imágenes de cada tipo de objeto y se ha realizado el proceso de segmentación, representación y descripción de cada una de estas. De esta manera el clasificador va aprendiendo que valores de cada patrón son representativos de cada clase de objetos. Hemos dividido el conjunto de imágenes en un conjunto de entrenamiento (70% de las imágenes) y en otro de test (30%). El de entrenamiento, como su nombre indica, se usará para entrenar al clasificador y el de test para probar la eficiencia de este.

Como clasificadores se van a utilizar KNN (K-Nearest Neighbors), DTREE (Decision Trees), SVM (Support Vector Machine), NN (Neural Networks), NBC (Normal Bayes Classifier), EM (Expectation-Maximization) y RTREE (Random Trees).

El usuario podrá seleccionar cualquiera de estos clasificadores desde un fichero de configuración.

ENTRENAMIENTO

Las matrices de entrenamiento son comunes para todos los clasificadores:

- *train* es la matriz que contiene los patrones de cada objeto. Si representamos cada objeto con siete patrones, suponiendo que tenemos N imágenes de entrenamiento, esta matriz es de tamaño Nx7.
- *classes* es una matriz de tamaño Nx1 que nos indica a que clase pertenece cada objeto. Por ejemplo si N=10 y hay 3 sartenes, 2 botellas y 5 vasos la matriz seria: [1 1 1 2 2 3 3 3 3 3]. El 1 representaría las sartenes, el 2 las botellas y el 3 los vasos.

Para entrenar los clasificadores se crea un objeto de su propia clase y se invoca el método *train*. Por ejemplo para la clase CvKNN sería de la siguiente forma:

```
CvKNN knn;
Knn.train(train,classes,….);
```

CLASIFICACIÓN

En la clasificación el algoritmo nos devuelve un número que indica la clase a la que pertenece la muestra:

```
float response = knn.find_nearest(muestra,…);
```

Siguiendo el ejemplo anterior, si response=2 el objeto sería una botella.

La muestra es una matriz de 1xN que contiene los N patrones que describen el objeto que queremos clasificar. Lo que hace esta función es buscar gracias al entrenamiento a que clase pertenece un objeto en concreto.

Una vez que sabemos que objetos hay en la imagen, escribimos sus nombres en la pantalla:

Figura 10.- Imagen con los objetos identificados.

4.4 SEGUIMIENTO DE OBJETOS (TRACKING)

Cuando ya hemos identificado los objetos que hay en la imagen y sabemos de qué objetos se tratan, es interesante seguir sus movimientos y saber donde se encuentran en todo momento, para ello utilizaremos el algoritmo de Lucas-Kanade.

En la fase de segmentación y descripción hallamos el contorno de los objetos. Para el seguimiento de los objetos vamos a utilizar el punto central de dicho

objeto. Para ello calculamos el rectángulo que contiene al contorno (véase figura 11) y a través de este es sencillo calcular el punto central, que es donde se cruzan las diagonales del rectángulo.

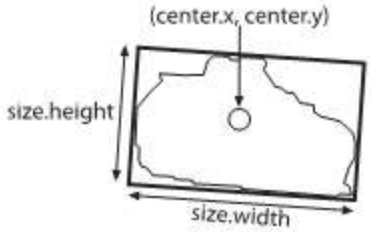


Figura 11.- Ejemplo del cálculo del centro de un contorno mediante el uso de el rectángulo que lo contiene.

La función CvRect r = **cvBoundingRect**(**c**) calcula el rectángulo *r* que contiene al contorno *c*.

Una vez que tenemos los puntos centrales de cada objeto el algoritmo se encarga de encontrarlos en los frames siguientes. Para ello, para cada frame que se sucede se calcula el flujo óptico usando de manera iterativa el método de Lucas-Kanade con imágenes piramidales. Esto se realiza mediante la función **cvCalcOpticalFlowPyrLK**, cuya misión es calcular las coordenadas del objeto en la posición actual a partir del frame anterior, del frame actual y de las coordenadas anteriores del objeto.

A continuación pintamos un punto en la imagen que se corresponde con las coordenadas del objeto.

Las figura 12 es un ejemplo del seguimiento de los objetos:

Figura 12.- La taza se mueve por la pantalla y en todo momento está identificada.

La posición (x,y) y el nombre de cada objeto se escriben en un archivo XML llamado ‘log.xml’ cada vez que se sucede un frame. De esta manera, al finalizar con la aplicación podemos ver en qué lugar ha estado cada objeto en todo momento.

4.5 BÚSQUEDA DE LAS MANOS EN LA IMAGEN

En este apartado se detallan los procedimientos que se han seguido para detectar la posición de las manos de la persona que manipula los objetos.

A diferencia del seguimiento de los objetos, explicado en el apartado anterior, este proceso no se basa en detectar las manos y luego seguirlas, sino que se deben detectar de nuevo en cada frame. Para ello se ha utilizado la búsqueda por histograma. Esta técnica se basa en la búsqueda de zonas en la imagen que tengan un color similar al de una imagen patrón. En nuestro caso, la imagen patrón contiene el color de la piel humana, por tanto, este algoritmo busca en la imagen zonas en las que aparezca este color para así localizar las manos.

Los pasos que se siguen cada vez que se recibe un frame son los siguientes:

- Conversión de la imagen patrón a formato HSV (matiz, saturación y brillo). Esta conversión es necesaria para poder realizar una búsqueda por histograma. La función que realiza esta conversión es **cvCvtColor**(**rgb, hsv, CV_BGR2HSV**) donde *rgb* es la imagen en formato RGB, *hsv* la imagen transformada a formato HSV y *CV_BGR2HSV* indica el tipo de conversión, en este caso RGB→HSV.
- Se obtiene el histograma de la imagen patrón. **cvCalcHist**(**hsv,hist**) es la función de OpenCV que calcula el histograma de una imagen *hsv* y lo almacena en *hist*.
- A continuación se transforma el frame a formato HSV de la forma explicada en el primer paso.
- Mediante la función **cvCalcBackProject**(**frame,result,hist**) buscamos en la imagen zonas que coincidan con el color de la imagen patrón. *frame* es la imagen sobre la que se busca, *result* es la imagen donde se almacenará el resultado y *hist* es el histograma de la imagen patrón. El resultado es una imagen en escala de grises que muestra las coincidencias: si los pixeles son negros significa que no ha encontrado coincidencias, si son blancos es que la coincidencia es perfecta y los grises intermedios son los diferentes grados de similitud.

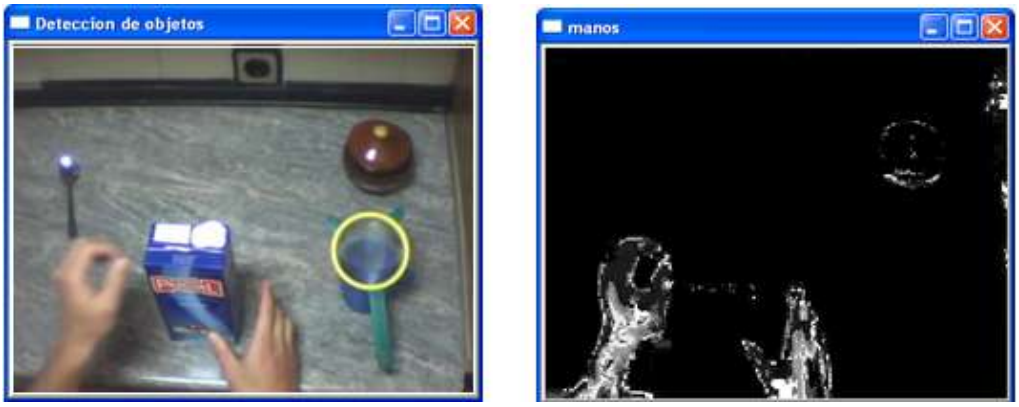


Figura 13.- La primera imagen es el frame recibido. La segunda imagen es el resultado de realizar la búsqueda por histograma del color de la carne.

- Como podemos ver en la figura 13 hay más zonas en la imagen aparte de las manos que tienen un color similar al de la carne humana. Para eliminar esas pequeñas zonas realizamos la operación morfológica de apertura. La función de OpenCV que realiza esta operación es **cvMorphologyEx**(**src,dst,temp,0,CV_MOP_OPEN,4**). Donde *src* es la imagen de entrada, *dst* es la imagen de salida, *temp* es una imagen temporal que utilizará el algoritmo, *0* indica que se usa el EE por defecto (cuadrado de 3x3 píxeles), *CV_MOP_OPEN* indica que se realiza la operación de apertura y *4* es el número de veces que se realiza esta operación.

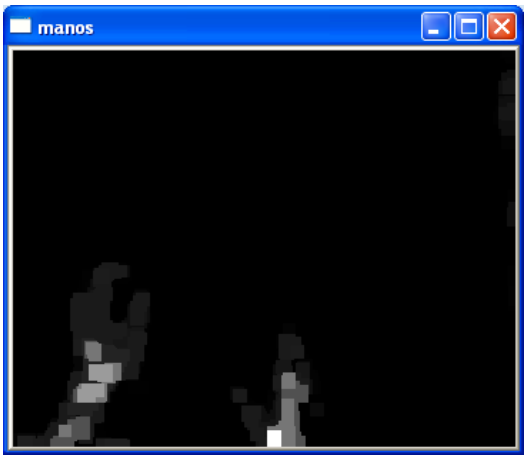


Figura 14.- Imagen resultante de realizar una apertura cuatro veces seguidas.

Como podemos ver en la imagen anterior tras realizar la apertura solo quedan las zonas más grandes, que coinciden con las manos.

- Para cerrar los huecos creados por la apertura en las manos realizamos la operación contraria, *el cierre*. Para llevar a cabo esta operación se usa la misma función que en el caso anterior pero indicando en este caso que se trata de un cierre: *CV_MOP_CLOSE* y el número de iteraciones, que es *20*, suficiente para cerrar todos los huecos.

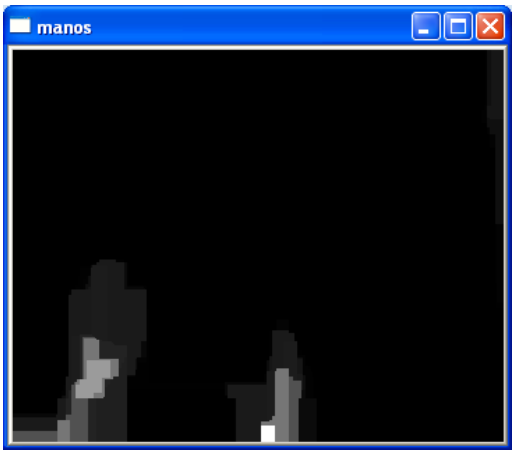


Figura 15.- Imagen resultante de realizar un cierre con 20 iteraciones.

- A continuación se pasa la imagen a formato binario mediante umbralización como se ha explicado en apartados anteriores:

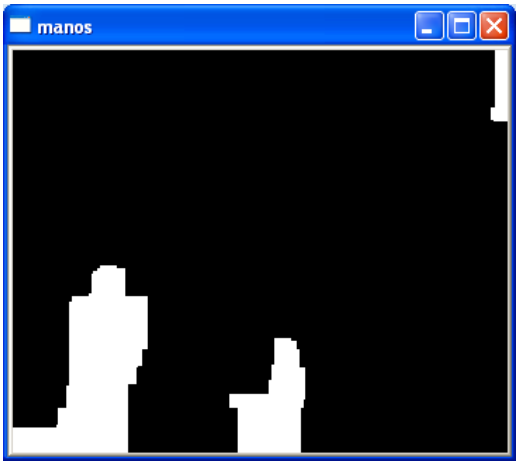


Figura 16.- Imagen resultante de una umbralización binaria. Todos los valores superiores a 20 pasan a ser 255 (blanco).

- El siguiente paso consiste en buscar los contornos de la imagen y posteriormente el centro de estos contornos con los procedimientos explicados en apartados anteriores. Luego se pintará en la imagen el centro de aquellos contornos que tengan un área mayor a 1000 pixeles.

Esto último es para eliminar zonas indeseadas como la que se puede ver en la parte superior derecha de la figura 16.

Figura 17.- Resultado final del proceso de búsqueda de manos.

CAPÍTULO V

Resultados obtenidos

5.1 PRESENTACIÓN DE LOS RESULTADOS OBTENIDOS

En este capítulo se dan a conocer los resultados obtenidos tanto en la clasificación y seguimiento de los objetos como en la detección de las manos.

RECONOCIMIENTO DE OBJETOS

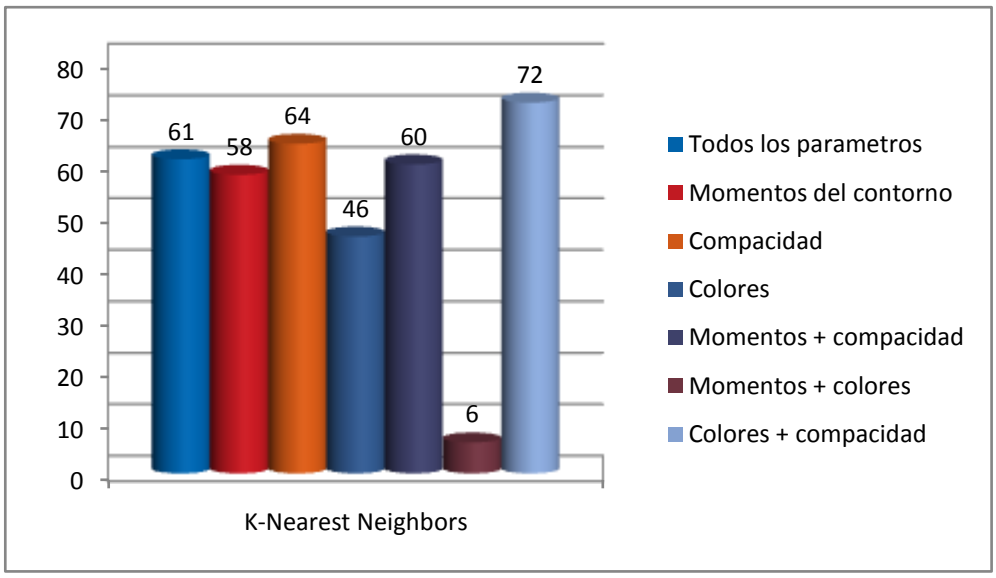
Para probar los clasificadores se dividieron las imágenes en un conjunto de entrenamiento y otro de test. Estos conjuntos estaban formados por 349 y 150 imágenes respectivamente, es decir, 70% de imágenes para entrenamiento y 30% para test.

Se han usado siete parámetros de descripción para cada objeto, que hemos dividido en tres grupos:

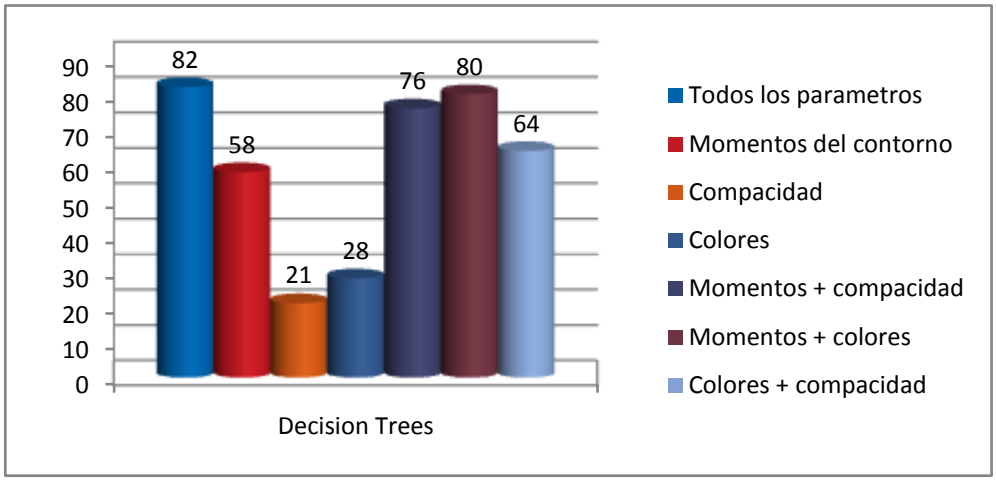
1. Momentos del contorno: incluye el momento espacial 00 y los centrales 02 y 20 (m_{00}, mu_{02}, mu_{20}).
2. Colores: incluye la media de los canales R, G y B del objeto.
3. Compacidad, que es el cociente entre el cuadrado del área y el perímetro.

Las siguientes gráficas presentan los resultados obtenidos para cada clasificador en función de los parámetros escogidos (porcentaje de acierto sobre las imágenes de test):

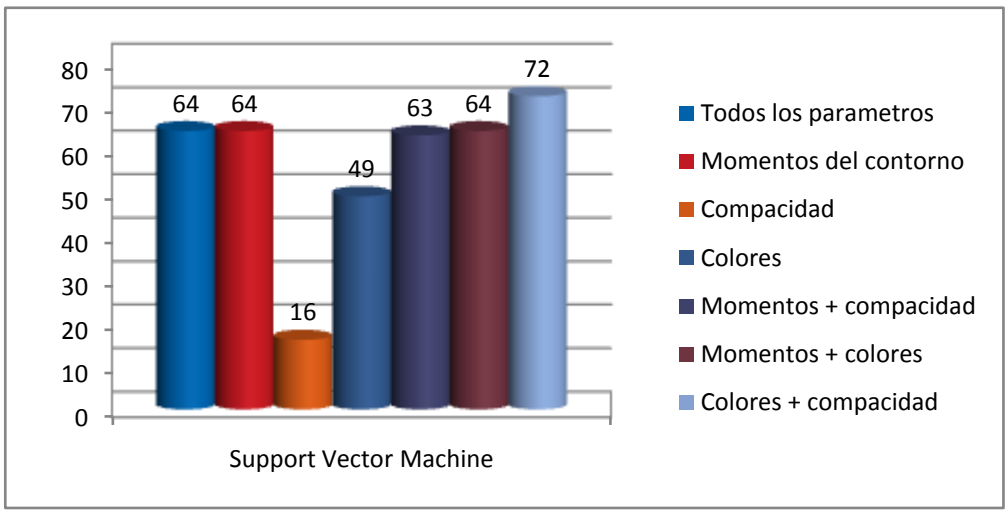
K-NEAREST NEIGHBORS



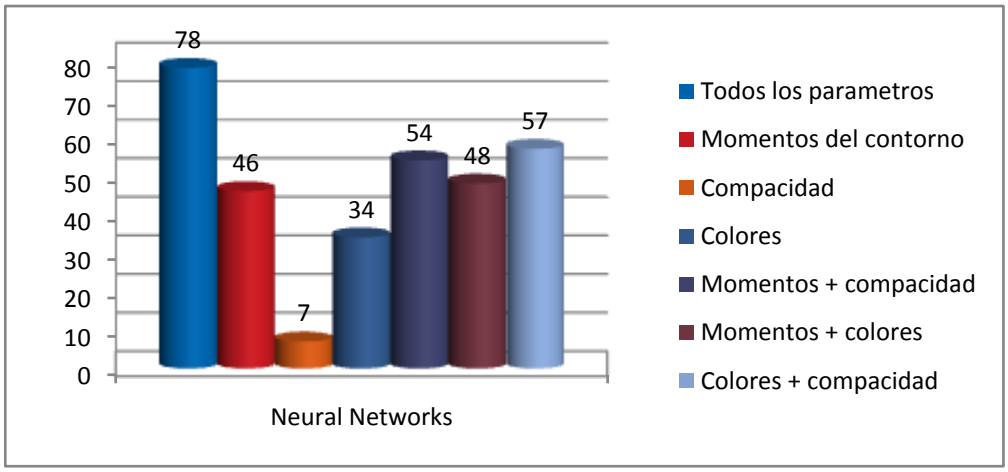
DECISION TREES



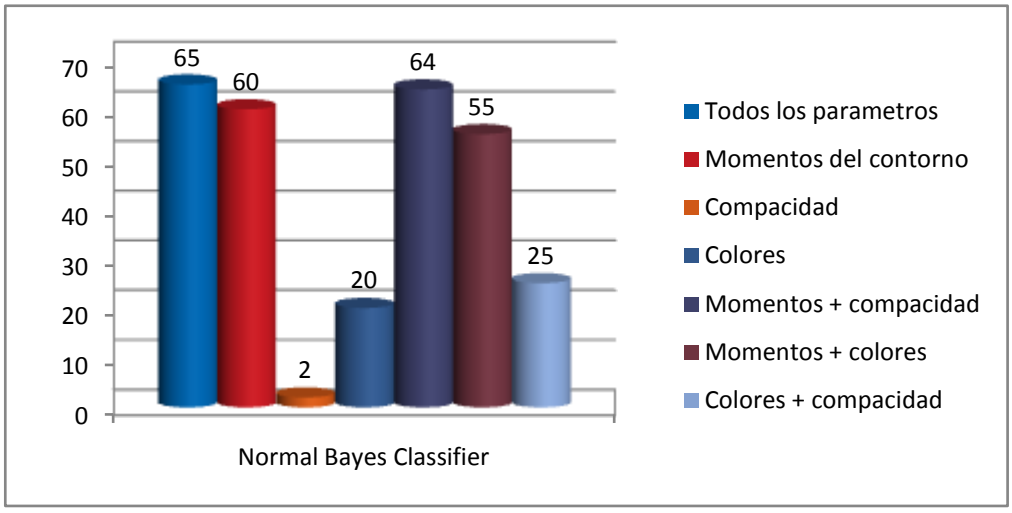
SUPPORT VECTOR MACHINE



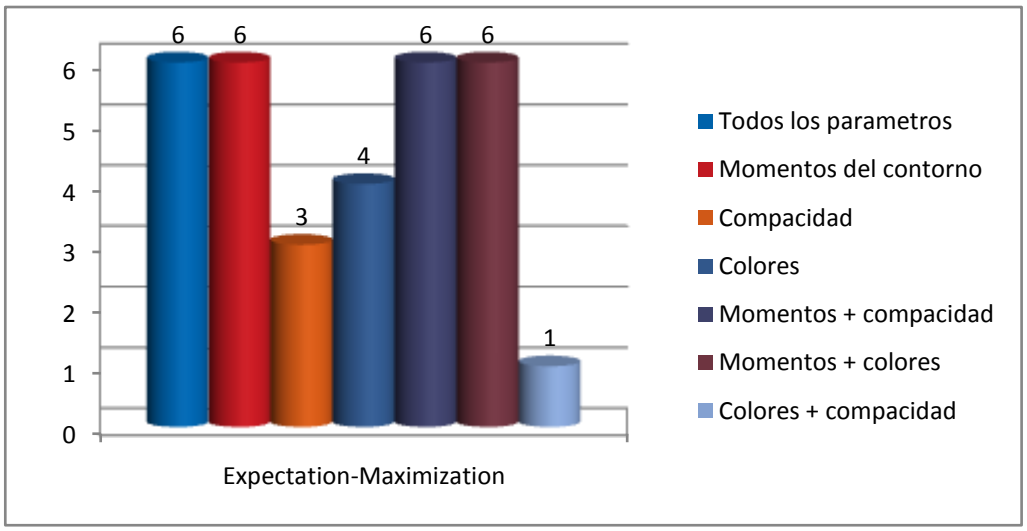
NEURAL NETWORKS



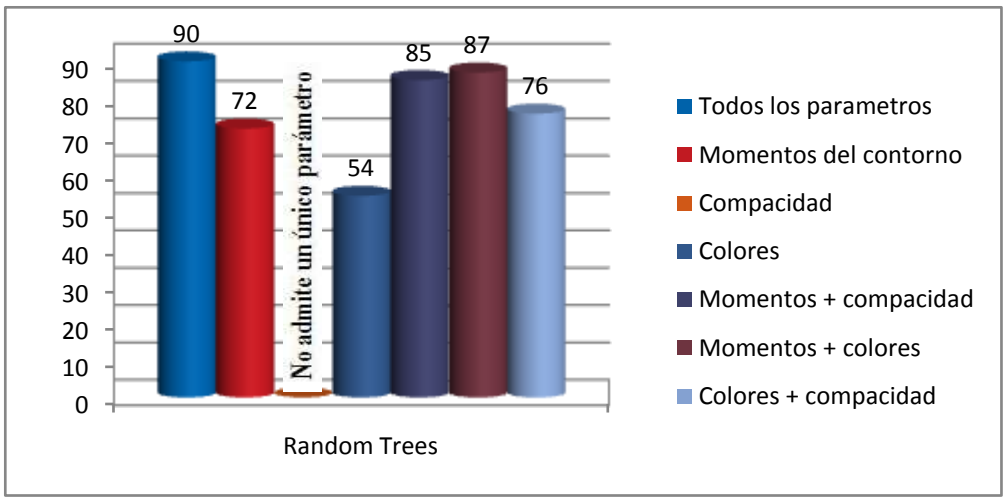
NORMAL BAYES CLASSIFIER



EXPECTATION-MAXIMIZATION



RANDOM TREES



Como podemos ver, no siempre por usar todos los parámetros se obtienen mejores resultados. Hay clasificadores que tienen un mayor porcentaje de acierto usando determinados parámetros debido a la forma en que opera su algoritmo.

Los resultados para el algoritmo *Expectation-Maximization* son bastante pobres, ya que, a diferencia de los demás clasificadores, es un algoritmo al que se le introducen los objetos de entrenamiento pero no se le dice a que clase pertenece cada objeto, sino que él mismo trata de averiguarlo.

Para el resto de algoritmos se obtienen buenos resultados (siempre por encima del 65% de acierto) usando alguna combinación de los parámetros. Por ejemplo, para el *Support Vector Machine* usando solo la compacidad obtenemos un 16% de acierto, pero combinando la compacidad con los colores obtenemos un 72%.

El algoritmo que mejor resultado ha dado son los arboles de decisión aleatorios llegando a un porcentaje de acierto del 90% usando todos los parámetros. Por lo tanto serán este clasificador y este conjunto de parámetros los elegidos para usarse por defecto en la aplicación.

SEGUIMIENTO DE OBJETOS

Las pruebas realizadas con el seguimiento de objetos han dado los siguientes resultados:

En objetos con un color liso y con una superficie poco reflectante el seguimiento es muy bueno. El objeto no se suele perder aunque el movimiento sea rápido e incluso en ocasiones, el objeto se pierde durante unos frames porque otro objeto lo tapa y al quedar de nuevo a la vista se vuelve a encontrar.

Figura 1.- Esta figura muestra como el vaso, que es un objeto con un color liso y no reflectante, llega a perderse en la imagen del medio pero al retirar el colador lo encuentra fácilmente.

El caso contrario son objetos muy reflectantes como tenedores, cuchillos, etc. Estos objetos son como espejos y al girarlos se refleja la luz u otros objetos en

ellos y cambia el color de los pixeles que lo componen, por lo que el programa no es capaz de encontrarlo en el siguiente frame:

Figura 2.- El cuchillo inicialmente está en una posición en la que no se refleja nada en él, pero al girarlo se refleja la luz y es mucho más claro. Al mover el cuchillo el algoritmo no es capaz de encontrar pixeles similares y lo pierde.

DETECCIÓN DE MANOS

El reconocimiento de las manos tiene unos resultados satisfactorios en general, pero tiene dos problemas importantes.

El primero y más importante es que ralentiza la aplicación, hace que los videos y la webcam vayan más despacio. Esto es debido a que necesita realizar muchas operaciones para cada frame. Aparte de buscar en la imagen zonas con el color de la piel, tiene que analizar todos los contornos para comprobar el área, además de las operaciones morfológicas citadas en el capitulo anterior. Por ello existe la opción de desactivar esta funcionalidad desde el fichero de configuración "entrenamiento.xml".

El segundo problema es que en ocasiones detecta manos en lugares donde no los hay debido a que son zonas con un color muy similar al de la piel.

5.2 CUMPLIMIENTO DE LOS OBJETIVOS INICIALES

Tras observar el capítulo anterior con los resultados obtenidos se puede concluir que se ha cumplido el objetivo inicial. Se ha conseguido crear una aplicación capaz de detectar objetos en una cocina, reconocer que tipo de objetos son y seguir sus movimientos.

Como hemos podido ver, la probabilidad de acierto en el mejor de los casos es bastante elevada, un 90%, y el seguimiento de los objetos es correcto salvo en casos excepcionales como los objetos reflectantes.

Incluso viendo que los resultados en un principio eran bastante prometedores, se decidió añadir varias opciones. Entre ellas cabe destacar: poder elegir entre varios clasificadores desde un fichero de configuración y la creación de un algoritmo para detectar las manos.

5.3 TRABAJOS FUTUROS

En este capítulo se describen cuales pueden ser las posibles mejoras a realizar, para intentar lograr mejores resultados en futuras aplicaciones que tomen este proyecto como punto de partida.

Para detectar los objetos es necesario que estos estén colocados de frente a la cámara. Se podría entrenar a los clasificadores para que también fueran capaces de reconocer los objetos colocados de distinta manera: tumbados, girados, al revés, etc. Sin embargo esto podría crear confusión a los clasificadores debido a que un mismo objeto se representa de múltiples formas y obtener peores resultados.

Uno de los principales problemas que hemos encontrado es la oclusión de los objetos por parte de la propia persona que los manipula o por otros objetos. Este problema se podría solucionar instalando varias cámaras de manera que la aplicación utilizara diferentes vistas del espacio de trabajo. Por lo tanto cuando un objeto tape a otro se podría utilizar otra cámara desde otro ángulo en el que esto no ocurra. Además se podría obtener la posición del objeto en 3D en lugar de 2D (x,y) como lo hace actualmente. Sería una aplicación de visión por computador en 3D.

Una cámara con un frame rate superior, más imágenes por segundo, podría mejorar el seguimiento de los objetos, ya que este falla cuando los objetos se mueven demasiado rápido.

CAPÍTULO VI

Bibliografía

6.1 REFERENCIAS BIBLIOGRÁFICAS

[AK89] Fundamentals of Digital Image Processing, Anil K. Jain, Ed. Prentice Hall, 1989.

[Bax94] G.A. Baxes, Digital Image Processing: Principles and Applications, J. Wiley & Sons, 1994.

[BK08] Learning OpenCV: Computer Vision with OpenCV, Gary Bradsky & Adrian Kaehler, O'Reilly, 2008

[Dai] http://www.dai.ed.ac.uk/HIPR2/hough.htm

[Esc01] A. de la Escalera, Visión por computador: Fundamentos y métodos, Pearson- Prentice Hall, 2001.

[Fra] Alexandre R.J. François; "MFSM: User Guide", http://mfsm.sourceforge.net/.

[GW93] R.C. González y R.E. Woods, Digital Image Processing, Addison Wesley, 1993.

[IMP] http://www.imageprocessingplace.com

[IVT] Integrating Vision Toolkit, http://ivt.sourceforge.net/.

[JVis] JavaVis, http://javavis.sourceforge.net/

[JKS95] R. Jain, R. Kasturi y B.G. Schunk, Machine Vision, McGraw-Hill, 1995.

[MAT] MATLAB, http://www.mathworks.com/products/matlab/.

[Open] Open Computer Vision Library Wiki, http://opencvlibrary.sourceforge.net/.

[Pra91] Digital Image Processing, William K. Pratt, 2nd ed, John Wiley & Sons, 1991.

[Ru95] The image processing handbook, John C. Russ, 2nd ed, CRC Press , cop. 1995.

[SHB99] M. Sonka, V. Hlavac y R. Boyle, Image Processing, Analysis and Machine Vision, PWS Publishing, 1999.

[Tec99] http://www.cs.technion.ac.il/Labs/Isl/Project/Projects_done/VisionClasses/Vision_1999/Hough/

[VeMo03] José Francisco Vélez Serrano, Ana Belén Moreno Díaz, Ángel Sánchez Calle, José Luis Esteban Sánchez-Marín. Visión por computador 2003.

[Wi] http:www.wikipedia.org/

ANEXO A

Manual de usuario

A.1 DESCRIPCIÓN GENERAL DEL SISTEMA

En este capítulo se describe la aplicación desarrollada, explicando de forma general cuales son las especificaciones del proyecto a nivel de usuario. En primer lugar, se realiza una descripción general del sistema para continuar con una descripción más profunda del funcionamiento de dicho sistema.

A.1.1 INTRODUCCIÓN

Este proyecto está enmarcado dentro del área de la visión por computador. El sistema desarrollado consiste en una herramienta de detención y seguimiento de objetos de cocina en tiempo real mediante el procesado de imágenes digitales.

A.1.2 ¿QUÉ HACE EL SISTEMA?

El sistema desarrollado, como se ha comentado en la introducción, se presenta al usuario como una herramienta que permite detectar los objetos que están presentes en las imágenes captadas por una webcam (o en su defecto en imágenes procedentes de un video capturado por esta). Una vez detectados, estos objetos pueden moverse de lugar. Otra de las funciones del sistema es averiguar en qué lugar se encuentran estos objetos en todo momento y escribir su posición frame a frame en un fichero XML. También existe la opción de poder detectar el lugar donde se encuentran las manos de la persona que manipula los objetos.

El sistema en un principio está diseñado y entrenado para la detección de objetos de cocina pero se puede entrenar para que funcione con cualquier tipo de objeto. Esta posible modificación no implicaría cambiar el código en ningún momento, únicamente se deberían tomar nuevas imágenes para entrenar la maquina y modificar un fichero de configuración en formato XML.

A.2 INSTALACIÓN Y EJECUCIÓN DEL SISTEMA

En este capítulo se presentan, de forma clara, los pasos a realizar por el usuario para instalar en su ordenador la aplicación desarrollada. Explicando paso a paso todas las acciones que tiene que llevar a cabo para lograr una correcta instalación del sistema y su posterior puesta en marcha.

2.2.1 REQUERIMIENTOS NECESARIOS

En este apartado se detallan los elementos software y hardware necesarios para la ejecución de la aplicación.

REQUERIMIENTOS SOFTWARE

Para la correcta ejecución de la aplicación es necesario tener instaladas las siguientes aplicaciones:

- Cygwin, esta aplicación podemos obtenerla en http://www.cygwin.com/
- La librería de visión por computador OpenCV 1.1, que podemos descargarla del siguiente sitio web: http://sourceforge.net/projects/opencvlibrary/

REQUERIMIENTOS HARDWARE

Para la correcta ejecución de la aplicación es necesario disponer de los siguientes elementos hardware:

REQUERIMIENTOS MÍNIMOS

- Ordenador Pentium III o superior.
- 512 Megabytes de memoria RAM
- Espacio libre en el disco duro de 1.5GB
- Tarjeta de video de 128 MB.
- Monitor con 640x480 píxeles de resolución
- CD-ROM
- Sistema operativo Windows XP SP2
- Webcam capaz de capturar a una resolución de 352x288.
- USB 1.1

REQUERIMIENTOS RECOMENDADOS

- Ordenador Pentium IV o superior.
- 1 GB de memoria RAM
- Espacio libre en el disco duro de 1.5GB
- Tarjeta de video de 256 MB.
- Monitor con 1024x768 píxeles de resolución
- CD-ROM
- Sistema operativo Windows XP SP3 o Windows Vista
- Webcam capaz de capturar a una resolución de 352x288.

- USB 2.0

2.2.3 INSTALACIÓN DEL SISTEMA EN EL DISCO DURO

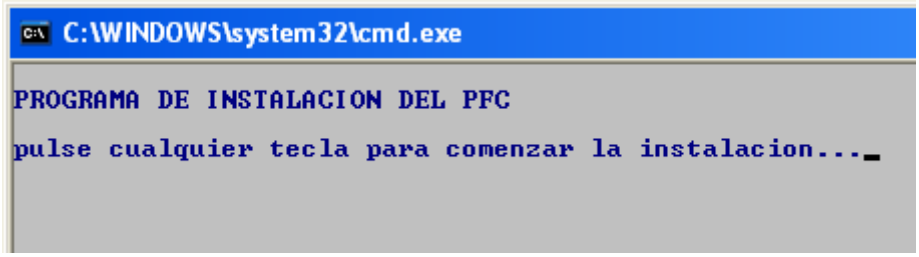
Para la correcta instalación de la aplicación, simplemente es necesario ejecutar el archivo "install.bat" que se encuentra en la carpeta llamada "Aplicación" que se acompaña con este proyecto. Para llevar a cabo esta tarea una vez hemos arrancado el ordenador y estando en el escritorio de Windows, hay que realizar los siguientes pasos:

- Introducir el CD que se acompaña con el proyecto en la unidad de CD-ROM del ordenador.
- Hacer doble clic en el icono con nombre "Mi PC".

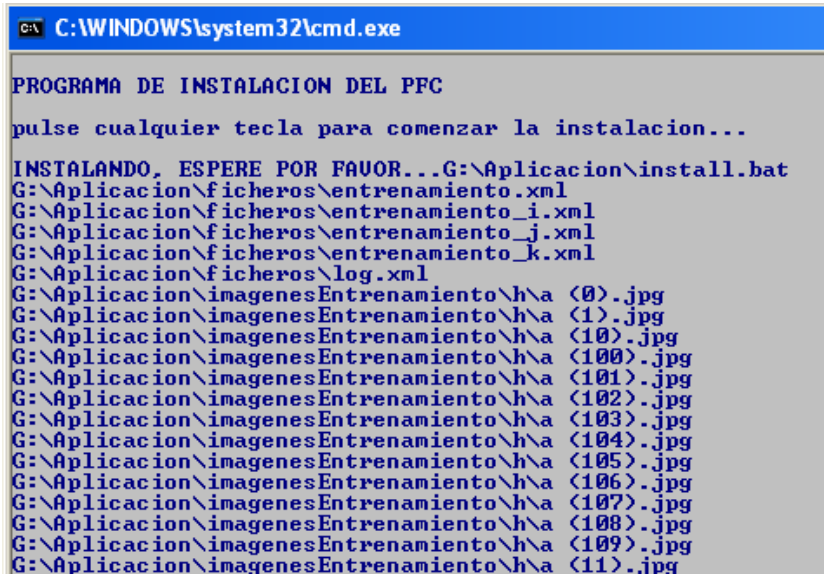
- Hacer doble clic en el icono de la unidad del CD-ROM, normalmente será la D.

- Abrir la carpeta con nombre "Aplicación" y ejecutar el archivo "install.bat".

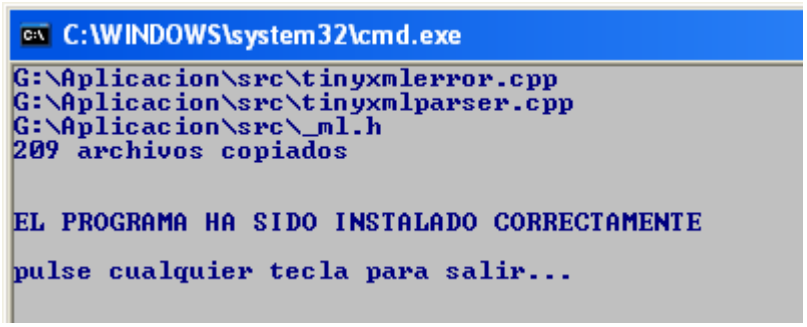
- A continuación se abrirá una ventana de MS-DOS:



- Pulsando cualquier tecla comenzará la instalación:



- Una vez finalizada la instalación aparecerá el siguiente mensaje:

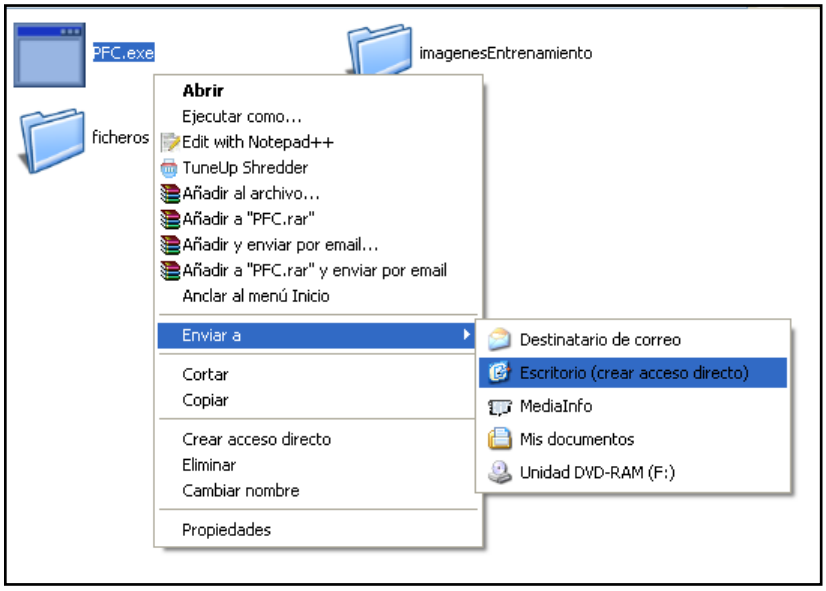


- Pulsamos cualquier tecla y la instalación ha finalizado. En caso de que no haya finalizado correctamente comprobar si se cumplen los pasos de los requisitos software.

Una vez realizado todos estos pasos tenemos perfectamente instalado en nuestro ordenador la aplicación y podemos pasar a ejecutarla.

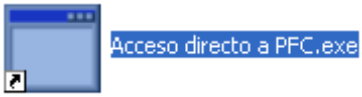
La instalación se ha realizado en el escritorio, se ha creado una carpeta llamada "Aplicación" en la que se encuentra el ejecutable "PFC.exe". Se puede crear un acceso directo del fichero ejecutable en el escritorio de Windows. Para ello una vez instalado la aplicación habría que:

- Abrir la carpeta que se acaba de generar en el escritorio, situar el icono del ratón encina del icono del fichero ejecutable "PFC.exe", y pulsar el botón derecho del ratón, apareciendo un menú desplegable.
- A continuación seleccionamos la opción "enviar a" y en el nuevo menú desplegable seleccionamos "Escritorio (crear acceso directo)". Dicha opción crea un acceso directo del fichero ejecutable en el escritorio de Windows. Pudiendo arrancar la aplicación siempre desde aquí y permitiéndonos olvidar la ruta de acceso a la carpeta donde almacenamos la aplicación.



2.2.4 EJECUCIÓN DE LA APLICACIÓN

Si hemos creado el acceso directo en el escritorio, la ejecución de la aplicación se reduce a situar el cursor sobre dicho icono y hacer doble clic con el botón izquierdo del ratón. Arrancando el entorno de la aplicación.



En caso contrario, para llevar a cabo la puesta en marcha del sistema hay que situarse nuevamente en la carpeta que se genero en el escritorio y hacer doble clic en ella para abrirla. Dentro de ella hacemos doble clic en el icono del fichero con nombre “PFC.exe”, arrancando la aplicación y apareciendo en la pantalla el entorno de la misma.

A.3 GUÍA DEL OPERADOR DEL SISTEMA

En este capítulo se explica al posible operador del sistema como se trabaja con el entorno de la aplicación describiendo detalladamente cómo funcionan cada una de las opciones presentes en la aplicación, los errores que pueden aparecer, las posibles modificaciones del sistema y la base de datos de objetos para los que la máquina se ha entrenado.

2.3.1 NOCIONES BÁSICAS

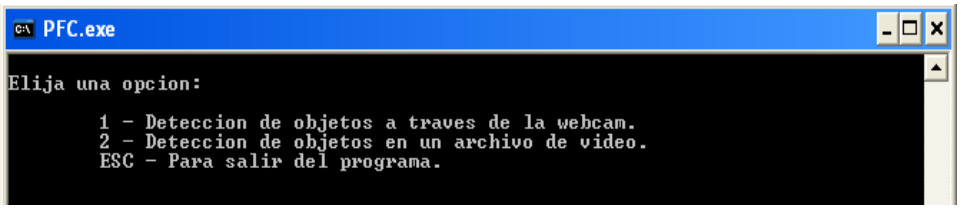
Para la utilización de la aplicación solo es necesario saber manejar el ratón y el teclado del ordenador, ya que toda la interacción aplicación-operador del sistema se realiza a través de estos dos periféricos del PC. Tampoco es necesario tener unos altos conocimientos dentro del área de la visión por computador, ya que la aplicación está diseñada y desarrollada para que la pueda utilizar cualquier persona.

EL ENTORNO DE TRABAJO

El entorno de la aplicación se compone de dos elementos:

- La ventana grafica, donde se mostrarán las imágenes captadas por la webcam o las imágenes de los videos:

- La consola, donde aparecerán las opciones que tiene el usuario en todo momento:



En la consola también se podrán escribir algunos datos que pida el programa, como el nombre del video a analizar.

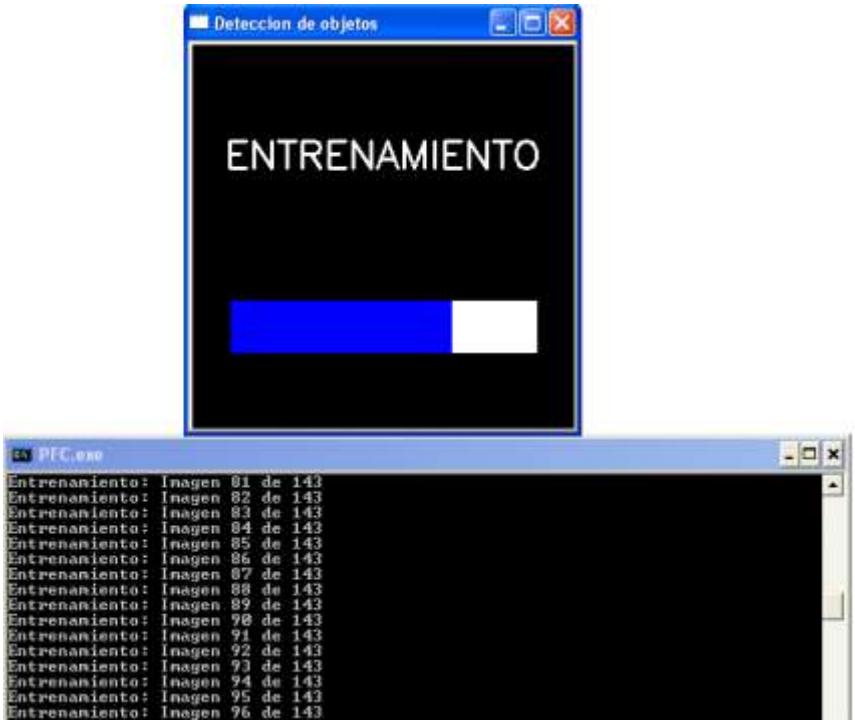
2.3.2 EXPLICACIÓN DE TODAS LAS OPCIONES DEL SISTEMA

Para este apartado vamos a suponer que tenemos la cámara instalada en un lugar fijo, enfocando siempre al mismo lugar que es donde se ha entrenado a la maquina. En el siguiente apartado llamado “2.3.3 Posibles modificaciones del

sistema" se explicará cómo conseguir que la maquina reconozca nuevos objetos que no estén en la base de datos o como cambiar la cámara de lugar.

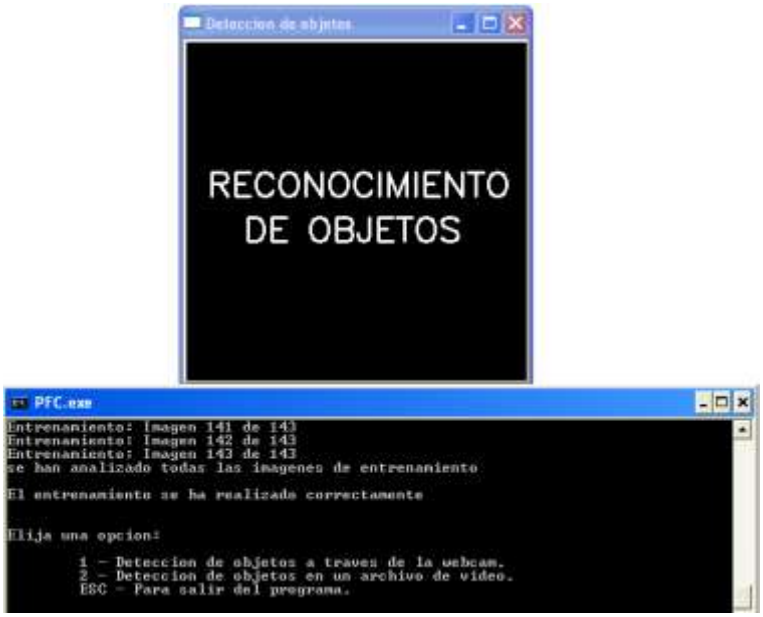
Una vez ejecutado el sistema aparece el entorno de la aplicación, que como hemos explicado en el apartado anterior consta de la ventana grafica y la consola:

Para empezar a entrenar la maquina basta con seleccionar la ventana grafica (esto es importante ya que si no la aplicación no hará nada) y pulsar cualquier tecla. En ese instante comenzará el entrenamiento:



La barra de progreso de la ventana grafica comenzará a completarse a medida que las imágenes de entrenamiento se van procesando. En la consola se mostrará el número de imagen que se analiza en cada instante.

Una vez finalizado este proceso aparecerá una nueva ventana:



Llegados a este punto podemos elegir entre tres opciones (primero debemos seleccionar la ventana grafica):

- Si lo que queremos es salir del programa pulsaremos la tecla ‘ESC’ y este se cerrará.
- Si pulsamos la tecla ‘1’ elegiremos detectar los objetos directamente mediante la webcam. Una vez pulsada esta tecla, si la webcam es correcta, automáticamente se tomará una imagen de los objetos y esta será analizada por el sistema apareciendo el nombre de los objetos en la imagen:

Ahora ya tenemos los objetos identificados y clasificados. Si pulsamos cualquier tecla (seleccionando primero la ventana grafica) pasaremos al seguimiento de los objetos. Podemos mover los objetos dentro del área de visión y estos serán seguidos por la aplicación:

Cada vez que la webcam captura un frame se escribe en un fichero XML el timestamp de ese frame, los objetos que aparecen en la imagen y su posición en píxeles. Este fichero se llama "log.xml" y se encuentra en la carpeta "Aplicación/ficheros/":

```
- <frame timestamp="12:46:17.593" n_frame="1">
    <objeto nombre="Bandeja del horno" x="221" y="105"/>
  </frame>
- <frame timestamp="12:46:17.640" n_frame="2">
    <objeto nombre="Bandeja del horno" x="221" y="105"/>
    <objeto nombre="Tabla" x="65" y="165"/>
  </frame>
- <frame timestamp="12:46:17.687" n_frame="3">
    <objeto nombre="Bandeja del horno" x="221" y="105"/>
    <objeto nombre="Tabla" x="65" y="165"/>
    <objeto nombre="Vaso" x="267" y="208"/>
  </frame>
- <frame timestamp="12:46:17.734" n_frame="4">
    <objeto nombre="Bandeja del horno" x="221" y="105"/>
    <objeto nombre="Tabla" x="65" y="165"/>
    <objeto nombre="Vaso" x="267" y="208"/>
  </frame>
```

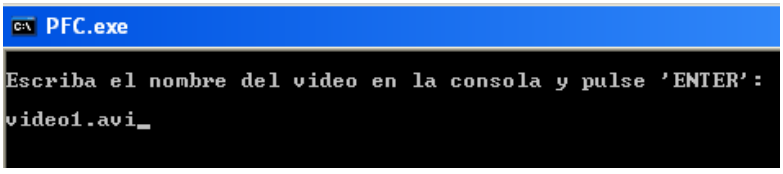
Además de este fichero también se crea un "log.txt" similar pero en formato de texto plano:

```
12:46:17.593
    Bandeja del horno (221,105)
12:46:17.640
    Bandeja del horno (221,105)
    Tabla (65,165)
12:46:17.687
    Bandeja del horno (221,105)
    Tabla (65,165)
    Vaso (267,208)
12:46:17.734
    Bandeja del horno (221,105)
    Tabla (65,165)
    Vaso (267,208)
12:46:17.781
    Bandeja del horno (221,105)
    Tabla (65,165)
    Vaso (267,208)
```

Si queremos que desaparezcan los puntos verdes que aparecen sobre los objetos y los siguen debemos pulsar la tecla '1', si queremos volver a la pantalla de inicio donde se nos preguntaba si queríamos analizar un video o directamente la webcam, pulsaremos '2', y por ultimo si lo que queremos es cerrar el programa pulsaremos la tecla 'ESC'.

Todos los frames capturados por la webcam se almacenarán en una carpeta llamada "Frames" dentro de la carpeta "Aplicacion".

- Si ya habíamos grabado un video con la webcam y queremos analizarlo debemos pulsar la tecla '2'.



A continuación nos situamos sobre la consola, escribimos el nombre del video y pulsamos 'ENTER':

Si el video existe, en la consola aparecerá el mensaje: "EL VIDEO ES CORRECTO", en caso contrario se nos indicará mediante la consola y podremos volver a introducir el nombre del video.
Llegados a este punto, la aplicación funciona de la misma manera que en el caso de la webcam. Para la aplicación, las imágenes del video son como si la webcam estuviese captando imágenes en el momento.

A la hora de usar el modo video, estos deben estar guardados en la carpeta "Aplicación/imagenesEntrenamiento/" y deben estar codificados sin compresión para que se puedan analizar correctamente las imágenes.

2.3.3 POSIBLES MODIFICACIONES DEL SISTEMA

Podemos realizar varias modificaciones en el sistema como elegir de entre una lista el clasificador con el que se reconocerán los objetos, elegir los parámetros de descripción de los objetos, activar la detección de manos, añadir más objetos a la base de datos o cambiar la cámara de lugar.

ELEGIR CLASIFICADOR

Mediante el fichero de configuración llamado "entrenamiento.xml" que se encuentra en la carpeta "ficheros" dentro de la carpeta "Aplicación" podemos elegir que clasificador usar:

```
<!--
   Clasificadores:
          1 - K-NEAREST NEIGHBORS
          2 - DECISION TREES
          3 - SUPPORT VECTOR MACHINE
          4 - NEURAL NETWORKS
          5 - NORMAL BAYES CLASSIFIER
          6 - EXPECTATION-MAXIMIZATION
          7 - RANDOM TREES
 -->
 <clasificador>4</clasificador>
<!--
```

Escribimos el número del clasificador entre las etiquetas <clasificador>X</clasificador>, guardamos el fichero y volvemos a ejecutar la aplicación. En este ejemplo estaríamos usando el *Neural Networks*.

SELECCIONAR PARÁMETROS DE DESCRPCIÓN DE LOS OBJETOS

En un principio para la realización del entrenamiento y clasificación de los objetos se usan los siguientes parámetros de descripción de los objetos:

- Momento central 20 del contorno: *momCentral.*
- Momento central 02 del contorno: *momCentral2.*
- Momento espacial 00 del contorno: *momEspacial.*
- Compacidad (perimetro^2/área): *compacidad.*
- Valor medio del canal R (rojo) de la imagen del objeto: *rMedio.*
- Valor medio del canal G (verde) de la imagen del objeto: *gMedio.*
- Valor medio del canal B (azul) de la imagen del objeto: *BMedio.*

Si por algún motivo se desea solo utilizar uno o varios de ellos, esto es posible modificando el fichero "entrenamiento.xml" que hemos citado anteriormente.

```
<!--Si incluir="1" se incluye ese parametro
    de descripcion del objeto en el reconocimiento
    si es "0" no se incluye
-->
<momCentral incluir="1"/>
<momEspacial incluir="1"/>
<momCentral2 incluir="1"/>
<compacidad incluir="1"/>
<rAvg incluir="1"/>
<gAvg incluir="1"/>
<bAvg incluir="1"/>
```

Aquellos parámetros que queramos incluir tendrán como valor "1" y aquellos que no queramos tendrán como valor "0". Una vez modificado a nuestro gusto, guardamos el fichero y volvemos a ejecutar la aplicación.
¡Cuidado! El clasificador 7, *Random Trees*, no admite un único parámetro, como mínimo se deben seleccionar dos.

ACTIVAR O DESACTIVAR EL RECONOCIMIENTO DE MANOS

Mediante el fichero de configuración "entrenamiento.xml" podemos activar o desactivar esta función. Hay que tener la precaución de que cuando esta función está activada la aplicación puede ralentizarse.

```
<!--Si buscar_manos es "1" se activa el
    buscador de manos en la aplicacion
-->
<buscar_manos>1</buscar_manos>
```

Como en alguno de los casos anteriores, si queremos activar la opción deberemos escribir el valor "1", en caso contrario el "0". Posteriormente guardamos el archivo y reiniciamos la aplicación.

AÑADIR MAS OBJETOS A LA BASE DE DATOS

Si lo que queremos es entrenar a la máquina para que reconozca además de los objetos que existen en la base de datos otros que no estén, hay que seguir los siguientes pasos:

Para facilitar la explicación supongamos que queremos añadir un estropajo y una tostadora.

En primer lugar debemos colocar el estropajo en la zona donde apunta la cámara y sacar de 6 a 12 fotos del objeto en distintas posiciones con la webcam. A continuación, se repite la misma operación pero con la tostadora. El siguiente paso es entrar en la carpeta "Aplicación/imagenesEntrenamiento/". Como podemos observar el nombre de las imágenes que hay en esta carpeta sigue el siguiente formato: "a (x).jpg". Lo que debemos hacer es añadir las imágenes que hemos sacado a esta carpeta con este formato de nombre. Primero, las imágenes del estropajo todas juntas y después, las de la tostadora. Si la última imagen que había en esa carpeta tenia por nombre "a (y).jpg" nuestras imágenes tendrán por nombre "a (y+1).jpg", "a (y+2).jpg", etc.

Una vez que hemos guardado y renombrado las nuevas imágenes entramos en la carpeta "Aplicación/ficheros/" y abrimos el fichero "entrenamiento.xml". Si habíamos tomado 6 imágenes del estropajo y 9 de la tostadora deberemos añadir a este fichero las siguientes líneas después del último objeto:

```
<entrenamiento>

    …

<objeto id="18" fotos="8" nombre="Mayonesa"/>

    ➔ <objeto id="19" fotos="6" nombre="Estropajo"/>
    ➔ <objeto id="20" fotos="9" nombre="Tostadora"/>

</entrenamiento>
```

Por último guardamos el fichero. La próxima vez que ejecutemos la aplicación será capaz de reconocer, además de todos los objetos que antes reconocía, el estropajo y la tostadora.

Para este apartado suponemos que la cámara está fija en el mismo lugar donde se entrenó para los anteriores objetos.

CAMBIAR LA CÁMARA DE LUGAR

Si cambiamos la cámara de lugar lo más posible es que tenga un ángulo de visión respecto a los objetos diferente al que tenía antes, por lo que probablemente, las imágenes que hay en la base de datos para entrenar el

sistema ya no serían validas. En ese caso, podemos volver a capturar imágenes desde la nueva posición de la cámara para entrenar nuevamente, de la misma manera que hicimos en el apartado anterior. Para no eliminar las imágenes anteriores por si algún día queremos volver a colocar la cámara en la posición original podemos hacer una nueva carpeta dentro de "Aplicación/imágenesEntrenamiento/" y en ella, guardar las nuevas imágenes que hemos capturado con el formato de nombre que ya hemos explicado en el apartado anterior. En este caso se empezaría desde la "a (0).jpg". También tenemos que obtener una imagen del lugar donde apunta la cámara y en la que no aparezca ningún objeto de los que vamos a analizar y a la que llamaremos "FONDO_ENTRENAMIENTO2.jpg", que se guardará también en la carpeta que hemos creado. Una vez realizados estos pasos iremos a la carpeta "Aplicación/ficheros/" y renombraremos el archivo que se llama "entrenamiento.xml", por ejemplo como "entrenamiento_A.xml", a continuación creamos un fichero nuevo con el nombre "entrenamiento.xml". Este fichero tendrá la misma estructura que el anterior pero cambiando lo siguiente:

- En esta línea indicaremos la carpeta donde hemos guardado las nuevas imágenes:
  <carpeta>**imagenesEntrenamiento/nuevaCarpeta**</carpeta>.
- En esta línea pondremos el numero de objetos que el sistema va a ser capaz de identificar: <numero_objetos>**18**</numero_objetos>.
- En las siguientes líneas procederemos como explicamos en el apartado anterior, indicando nombre y número de fotos de cada objeto.
  <objeto id="**1**" fotos="**8**" nombre="**Cuenco**"/>. El orden en que se añade cada objeto es el mismo orden en que se guardaron las imágenes, si no se hace así, la aplicación se confundirá a la hora de reconocer los objetos.

Para terminar guardamos el fichero. Al abrir de nuevo la aplicación, esta se entrenará con las nuevas imágenes automáticamente y será capaz de reconocer objetos desde otra posición.
Si quisiéramos volver a la configuración anterior solo tendríamos que restablecer el fichero "entrenamiento.xml" como estaba en un principio.

2.3.4 POSIBLES ERRORES EN EL SISTEMA

Existen varios errores que se pueden producir a la hora de trabajar con la aplicación, en este apartado trataremos de explicarlos:

- La cámara debe estar fija en un sitio sin moverse, ya que la aplicación para detectar los objetos calcula la diferencia entre una imagen con los objetos y otra sin ellos. La imagen con los objetos es la que se captura en el momento y la imagen del fondo está guardada en la base de datos,

con lo cual, si la cámara se mueve, la aplicación no detectaría de manera correcta los objetos.

- Si en el lugar donde apunta la cámara hay poca luz o sombras muy pronunciadas la aplicación tendría muchas dificultades para detectar los objetos.
- Cuando se está produciendo el seguimiento de los objetos es importante no tapar los objetos ya que el programa los perderá de vista y no sabrá donde se ubican.
- Para que, a la hora de seguir los objetos, la aplicación no los pierda es importante no moverlos a una velocidad excesiva.

2.3.5 OBJETOS ENTRENADOS

Los objetos para los que la maquina ha sido entrenada son los siguientes:

Taper

Cazo

Taza

Sartén

Tijeras de la carne

Salero

Vaso

Frasco de especias

Cafetera

Kétchup

Plato

Embudo

Azucarero

Mayonesa

Mostaza

Bandeja de horno

Brick de leche

Botella

Tabla de cortar

Sacacorchos

Colador de tela

Cuenco

Jarra

Cacerola

Pinzas de la barbacoa

Untador de paté

Cuchara

Cuchara de madera

Cazo sopero

Espumadera

Cuchillo de la carne

Hacha de la carne

Cuchillo del jamón

Tenedor

Cuchillo

Cucharilla